VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV VÝKONOVÉ ELEKTROTECHNIKY A ELEKTRONIKY
DEPARTMENT OF POWER ELECTRICAL AND ELECTRONIC ENGINEERING

OPTIMALIZACE KONSTRUKCE ELEKTRICKÝCH STROJŮ POMOCÍ GENETICKÝCH ALGORITMŮ
OPTIMIZING DESIGN OF ELECTRIC MACHINES USING GENETIC ALGORITHMS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — Marek Jelének
AUTHOR

VEDOUCÍ PRÁCE — Ing. Marcel Janda, Ph.D.
SUPERVISOR

BRNO 2016

VYSOKÉ UČENÍ
TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

Ústav výkonové elektrotechniky a elektroniky

Bakalářská práce

bakalářský studijní obor
Silnoproudá elektrotechnika a elektroenergetika

Student: Marek Jelének
Ročník: 3

ID: 159249
Akademický rok: 2015/16

NÁZEV TÉMATU:

Optimalizace konstrukce elektrických strojů pomocí genetických algoritmů

POKYNY PRO VYPRACOVÁNÍ:

1. Seznamte se s optimalizačními metodami použitelnými v oblasti elektrických strojů.
2. Navrhněte postup využití optimalizačních metod v návrhu konstrukce elektrických strojů.
3. Vytvořte jednoduchý optimalizační algoritmus a aplikujte ho na jednoduchý příklad.

DOPORUČENÁ LITERATURA:

Dle doporučení vedoucího.

Termín zadání: 21. 9. 2015

Termín odevzdání: 31. 5. 2016

Vedoucí práce: Ing. Marcel Janda, Ph.D.
Konzultanti bakalářské práce:

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ Fakulta elektrotechniky a komunikačních technologií Ústav elektroenergetiky

doc. Ing. Petr Toman, Ph.D.
předseda oborové rady

UPOZORNĚNÍ:

Abstrakt

Tato práce se zabývá optimalizačními metodami použitelnými v oblasti konstrukce elektrických strojů. Hlavním cílem práce je navrhnutí optimalizační metody, její aplikování na jednoduchý příklad a zhodnocení výsledku před a po optimalizaci.

Abstract

This bachelor thesis deals with optimization methods usable in the construction of electrical machines. The main aim of this thesis is propose an optimization method, applied to simple example and evaluate optimized results with not optimized results.

Klíčová slova

Optimalizace; algoritmus; funkce; elektrický stroj; extrém; globální; lokální; minimum; maximum; prohledávaný prostor; rozběhový proud; moment;

Keywords

Optimization; algorithm; function; electric machine; extreme; global; local; minimum; maximum; search space; starting current; torque;

Bibliografická citace

JELÉNEK, M. *Optimalizace konstrukce elektrických strojů pomocí genetických algoritmů.* Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2016. 48 s. Vedoucí semestrální práce Ing. Marcel Janda, Ph.D.

Prohlášení

Prohlašuji, že svou bakalářskou práci na téma Optimalizace konstrukce elektrických strojů pomocí genetických algoritmů jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené bakalářské práce dále prohlašuji, že v souvislosti s vytvořením této bakalářské práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení § 152 trestního zákona č. 140/1961 Sb.

V Brně dne …… Podpis autora ……..

Poděkování

Děkuji vedoucímu bakalářské práce Ing. Marcelu Jandovi Ph.D. za účinnou metodickou, pedagogickou a odbornou pomoc a další cenné rady při zpracování mé bakalářské práce.

V Brně dne …… Podpis autora ……..

OBSAH

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM GRAFŮ

SEZNAM SYMBOLŮ A ZKRATEK

SOA		Swarm Optimization Algorithms (optimalizační algoritmus hejna)
MOEA		Multiple Objective Evolutionary Algorithms (evoluční algoritmy více proměnných)
GA		Genetic Algorithm (genetické algoritmy)
SA		Simulated Annealing (simulované žíhání)
SQP		Sequential Quadratic Programming (sekvenční kvadratické programování)
$f(x)$		Fitness funkce
$h(x)$		Heuristická funkce
P		Pravděpodobnost
T	(-)	Koeficient reprezentující teplotu ve výpočtu P v SA
ΔE	(-)	Koeficient reprezentující rozdíl energií ve výpočtu P v SA
t	(s)	Doba výpočtu
n	(-)	Počet kroků výpočtu
x_0	(-)	Hodnota počátečního bodu na ose x
y_0	(-)	Hodnota počátečního bodu na ose y
τ	(-)	Feromon vylučovaný mravenci v algoritmu mravenčích kolonií
P1 – P6		Označení jedinců ve výchozí generaci v GA
P1’ – P6‘		Označení jedinců v nové generaci v GA
AIP		Autodesk Inventor Professional
AM		Ansoft Maxwell
AW		Ansys Workbench

ÚVOD

Během 21. století se značně zrychlil pokrok v celé řadě odvětví, jako je chemické, strojírenské, popřípadě IT a v oblasti elektrotechniky a elektrických strojů tomu není jinak. Právě pokrok v ostatních odvětvích umožnil preciznější konstrukce elektrických strojů. S výhodami se využívají materiály s větší mechanickou pevností, nebo s optimálnějšími magnetickými vlastnostmi, popřípadě i větší chemickou odolnosti. Dále také velký pokrok v oblasti EMC umožnil celou řadu nových aplikací. Abychom si uvědomili všechen pokrok, je dobré se ohlédnout zpět do historie, kde je patrný skok oproti technologii používané dnes. Od prvního pokusného elektromotoru v roce 1834 sestrojeného jistým Moritzem H. Jacobim, přes první koncepce stejnosměrného stroje od Wernera Siemense a Charlese Wheatstona, popřípadě asynchronního motoru od Nikoly Tesly až po nejnovější varianty všech elektrických strojů, jak je známe dnes [1]. Pomocí moderní výpočetní techniky a softwaru lze účinně řešit soustavy rovnic při analýze elektromagnetických, elektrostatických nebo například tepelných jevů. Zde nacházejí uplatnění optimalizační metody, pomocí nichž lze zdokonalovat již stávající zařízení nebo napomáhat při konstrukci zařízení nových. Optimalizačních metod je řada, jako například genetické algoritmy – GA, algoritmy simulovaného žíhání – SA nebo gradientní metoda, popřípadě metoda sekvenčního kvadratického programování – SQP, a mnohé další [2]. Kombinací těchto optimalizačních metod, výpočetní techniky a širokého spektra softwaru zabývajícího se touto problematikou lze docílit rychle a efektivně výsledků, které dříve zabíraly mnohonásobně více času. Je ovšem potřeba mít na paměti, že ani software není neomylný, a proto je na místě ověřit například měřením, nebo kontrolním výpočtem, že výsledky optimalizačních metod jsou důvěryhodné.

1 OPTIMALIZAČNÍ METODY

Pojmem optimalizace označujeme hledání nějakého optimálního řešení daného problému. Potřebujeme-li například naplánovat cestu na dovolenou, tak existuje řada možných tras, ale pro nás může být důležitá ta s nejmenší vzdáleností, nebo nejkratší dobou cesty, popřípadě místo skrze které chceme cestovat. Na základě těchto kritérií jsme pak schopni ze všech možných cest vybrat tu, která nejlépe splňuje požadavky. Pokud bychom tento problém převedly do oblasti elektrotechniky, tak se může jednat například o optimalizaci na co největší účinnosti motoru, nebo co nejmenší oteplení při různých stavech celé řady součástek nebo strojů respektive přístrojů. Pole, ve kterém lze aplikovat optimalizaci je značně rozsáhlé. Abychom byli schopni řešit optimalizování, tak je nutné popsat námi zkoumaný problém. Popisem se rozumí matematický popis. Nejprve provedeme analýzu problému a určíme jednotlivé prvky, ať již je to délka cesty, nebo doba jízdy, popřípadě výpočet oteplení potažmo účinnosti nebo jakékoliv jiné proměnné. Těm potom přiřadíme označení, nebo je popíšeme nějakou rovnicí.

Samotný proces optimalizace řeší nalezení maximální resp. minimální hodnoty zkoumané veličiny nebo systému více veličin. Tyto veličiny jsme obvykle schopni popsat a vypočítat, pak je označíme za vypočitatelné, v opačném případě jsou nevypočitatelné. Tato práce se bude zabývat pouze veličinami vypočitatelnými. Pokud jsou veličiny mezi sebou v určitém vztahu, například účinnost motoru je ovlivněna ztrátami ve vinutí rotoru, ztrátou ve vinutí statoru a řadou dalších veličin, pak lze vztah mezi těmito veličinami popsat funkcí a výsledkem optimalizace je pak optimum zkoumané funkce. Tato funkce má různá označení v závislosti na používané optimalizační metodě, ale matematicky se jedná vždy o tu samou funkci.

Můžeme se setkat například s označením heuristická funkce, kriteriální funkce, fitness funkce atd. [5]. Jako příklad takovéto funkce může být například převod binárního čísla A_2= 10010 na číslo desítkové soustavy A_{10}:

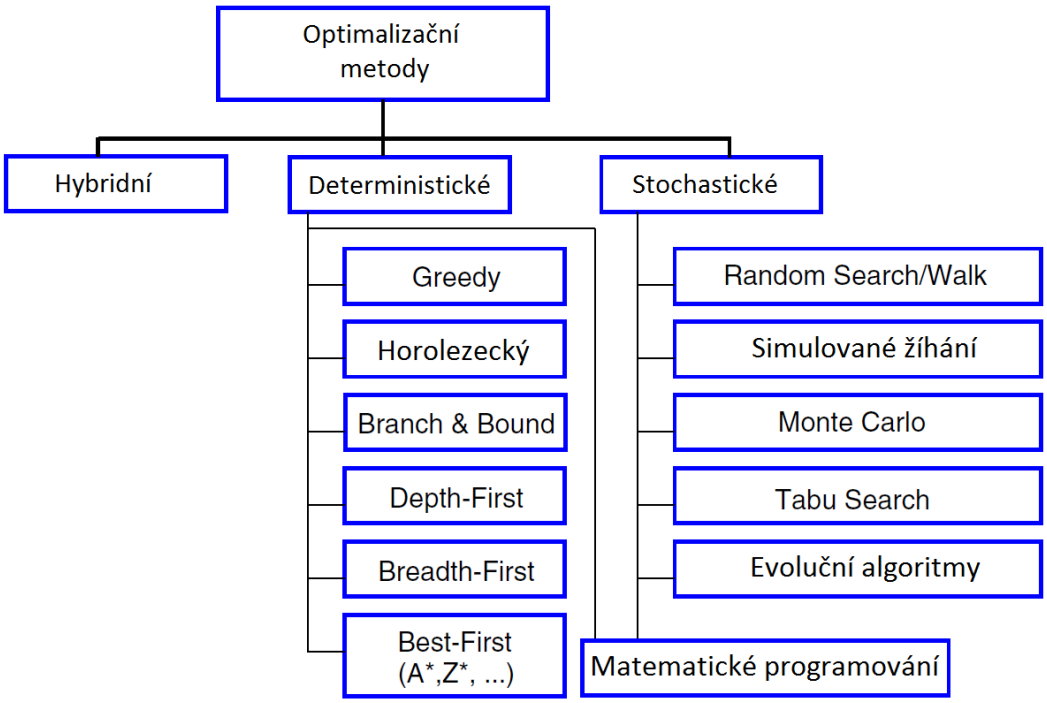
$$f(x_i) = x_i \cdot 2^{n-i} + x_i \cdot 2^{n-i} + \ldots + x_i \cdot 2^{n-i} \tag{1.1}$$

kde x_i je označení hodnoty jednotlivých bitů binárního čísla a n je celkový počet bitů.

$$f(x_i) = 1 \cdot 2^4 + 0 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 = 18 \tag{1.2}$$

V tomto jednoduchém příkladu můžeme x_i nazvat váhovým koeficientem a 2^{n-i} veličinou. Budeme – li optimalizovat tuto funkci na maximum, tak budeme hledat takové x_i, pro které je $f(x_i)$ největší. Analogicky k rovnici 1.1 můžeme sestavit libovolnou rovnici, ve které váhový koeficient x_i bude nabývat hodnot $x_i \in \langle 0,1 \rangle$, a tím bude určovat důležitost jednotlivých veličin, nebo můžeme použít jinou metodu, kterou mezi sebou svážeme jednotlivé veličiny. Samotné veličiny, jako například teplota, účinnost, intenzita elektrického pole atd. jsou pak vypočítávány různými způsoby, například metodou konečných prvků, měřením, nebo výpočtem z tabulkových hodnot [4] a stejně tak existuje řada optimalizačních metod, kterými lze hledat optimum.

Po určení veličin, nebo sestavení kriteriální funkce je na řadě volba optimalizační metody, které můžeme rozdělit do několika skupin podle způsobu procesu optimalizace. Nejzákladnějším, avšak dostatečným dělením je rozdělení do těchto skupin:



***Obr. 1** Dělení optimalizačních metod [10]*

Deterministické metody využívají matematického výpočtu a předpovědi dalšího kroku na základě výpočtu gradientů funkce, tedy se předpokládá, že existuje první a druhá derivace funkce) resp. známe matematický popis veličin. Při nalézání globálního extrému multimodální funkce, tj. funkce s mnoha extrémy, mívají tyto metody sklony konvergovat k extrému blízko startovního bodu a nejsou schopny tento bod opustit. Existují jejich různé modifikace, které se tomuto problému dokáží vyvarovat, avšak jsou komplikované a pro potřeby této práce nejsou stěžejní, a proto se jimi nebudu zabývat. Oproti tomu stochastické metody, jak již napovídá jejich název, hledají optimální řešení s jistým prvkem náhodného výběru a mnohem úspěšněji tak řeší multimodální typ funkcí. Hybridní metody pak využívají náhodného výběru počátečních bodů a následné řešení deterministickým způsobem [5].

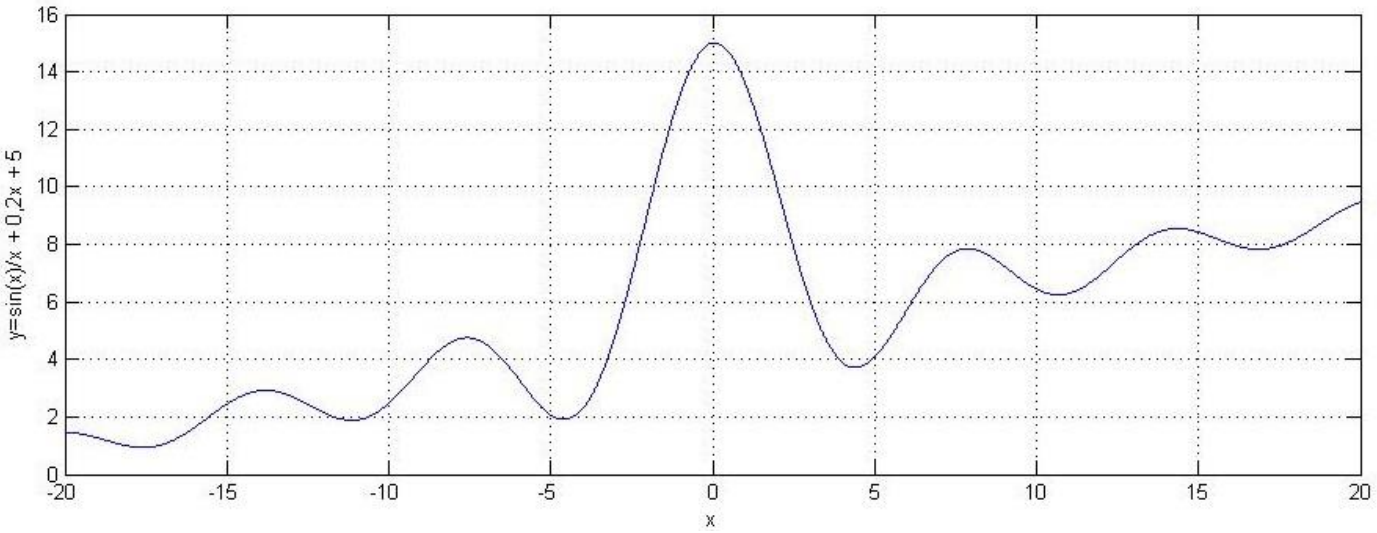
Protože u elektromagnetických typů úloh je obvyklé, že předem není znám průběh prohledávaného prostoru, respektive je pravděpodobné, že známe popis jednotlivých veličin, ale nevíme přesný popis jejich vzájemného působení, tak je výhodné použít stochastické metody. Kupříkladu dokážeme vypočítat proudy a napětí v konkrétních částech našeho zařízení, určit oteplení při různých stavech atd. ale nevíme, jakým způsobem změna těchto veličin bude ovlivňovat například účinnost. Z tohoto důvodu jsou pro jejich řešení vhodnější metody hledající globální extrémy, a to i bez znalosti podrobného matematického vyjádření problému. Zaměřuji se proto podrobněji na stochastické metody, které jsou v této oblasti úspěšnější. Jak již bylo zmíněno v úvodu, tak s rostoucím výkonem výpočetní techniky klesají nevýhody těchto metod, tedy časová náročnost způsobená vyšším počtem kroků [1]. I přes to však kompletní výpočet a analýza elektrického motoru je velice časově a výkonnostně náročná úloha. Důsledkem toho se často před zahájením optimalizace provádí tzv. citlivostní analýza, jejímž výsledkem je zjištění, změnou které veličiny nejvíce ovlivněn výsledek a naopak. Při optimalizaci se pak algoritmus

může soustředit na důležitější veličiny předně a tím zkrátit dobu potřebnou k nalezení optima. Připomínám, že optimum nemusí být pouze minimem, ale také maximem funkce.

Ke zvolení vhodné optimalizační metody pro konstrukci elektrických strojů je nejprve nutné porozumět jednotlivým optimalizačním metodám, a proto některým z nich bude v následujících kapitolách věnována zvýšená pozornost. U každé z vybraných metod se budu zabývat principem, na kterém jsou založeny a postupem, s jakým prohledávají zkoumaný prostor. Ten bude zjednodušen v blokovém schématu.

Aby bylo názornější, jakým způsobem probíhá optimalizace, tak pomocí každého algoritmu byla provedena optimalizace jednoduché funkce:

$$y = \frac{\sin(x)}{x} + 0{,}2x + 5 \tag{1.3}$$



Graf 1 *Graf znázorňující průběh rovnice 1.3*

Tuto funkci jsem si vybral, protože její průběh vhodně poslouží k demonstraci vlastností každé z uvažovaných optimalizačních metod. Pomocí všech algoritmů budu hledat maximální hodnotu této funkce. Jak je vidět v **Graf 1**, tak funkce má jeden globální extrém a řadu lokálních extrémů. U všech algoritmů provedu optimalizaci z několika různých bodů *x*, aby se prokázalo, jestli změna počátečního bodu má vliv na proces optimalizace, či nikoliv. Tyto hodnoty byly zvoleny následovně:

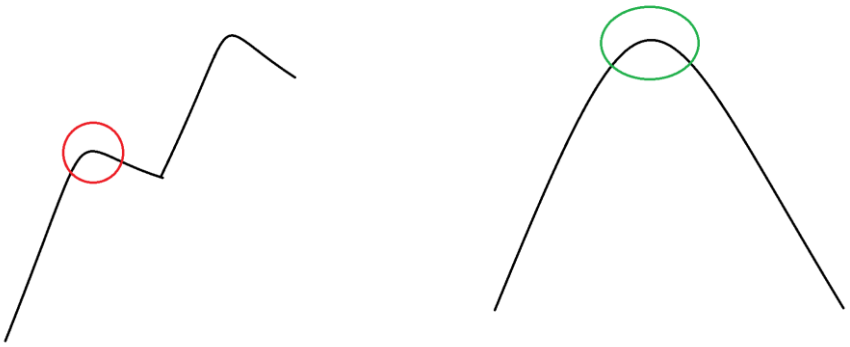
$$x_0 = \{-15, -10, -3, 3, 10{,}15\} \tag{1.4}$$

Optimalizace byly provedeny pomocí programu MATLAB a jednotlivé výsledky pro přehlednost porovnány v samostatných tabulkách.

1.1 Horolezecký algoritmus

Jak již napovídá název této optimalizační metody, tak její princip je založen na chování horolezce při snaze vyšplhat se na vrchol hory. Horolezec při výstupu postupuje hmat po hmatu, resp. krok po kroku a zhodnocuje, zda ho tento krok posune výše [3]. U horolezeckého algoritmu se ještě vyhodnocuje, který ze směrů má největší sklon, myšleno tak, aby byl horolezec u cíle co nejrychleji. Je zřejmé, že pro potřeby algoritmu se může jednat jak o výstup, tak i o sestup, tedy může jít o hledání minima nebo maxima v prohledávaném prostoru hodnot.

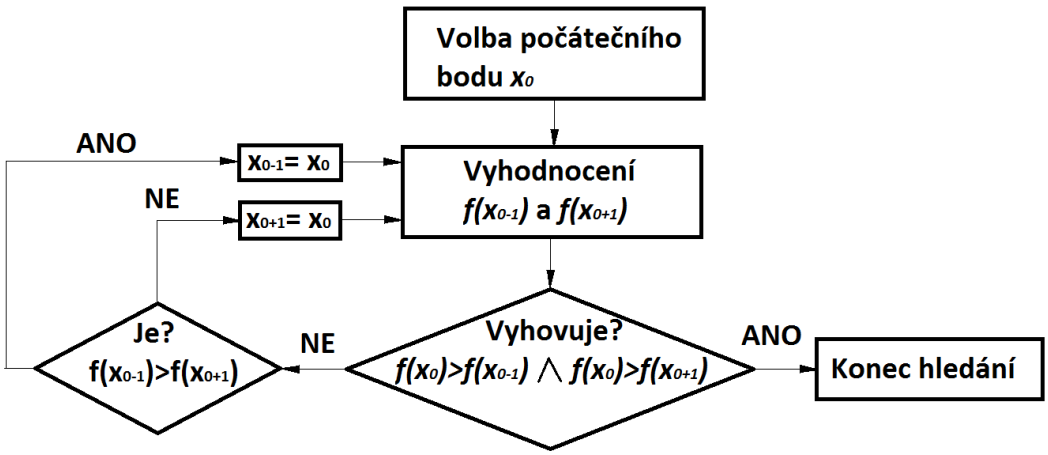
Samotný algoritmus je považován za jeden ze základních algoritmů pro optimalizaci, a proto jej zde uvádím i přes to, že je to algoritmus, který umožňuje hledání pouze lokálního extrému. Princip je takový, že prozkoumává hodnoty v okolí aktuálního bodu na základě heuristické funkce, což je funkce, která porovnává, jestli daná hodnota je vhodnější či nikoli, např. zda je horolezec výše nebo níže [3]. Tvar heuristické funkce může být různý. Na tvaru této funkce je však závislý výsledek samotného algoritmu. Bude-li mít funkce tvar, jako je na **Obr. 2a**, a budeme-li hledat maximum této funkce, tak záleží na počátečním bodě, ze kterého bude algoritmus začínat. Je pravděpodobné, že algoritmus vyhodnotí jako optimální řešení pouze lokální extrém. Pokud ovšem bude mít funkce tvar jako je na **Obr. 2b**, a budeme-li opět hledat maximum, tak nehledě na počátečním bodě vždy dosáhneme globálního maxima jako optimálního řešení.



a) lokální extrém b) globální extrém

***Obr. 2** Extrémy funkce*

Jestli se jedná o globální nebo lokální extrém samotný algoritmus nezjistí. Horolezecký algoritmus je vhodný hlavně pro rostoucí nebo klesající funkce s jedním lokálním, tedy i zároveň globálním extrémem. Pokud využijeme hybridní metodu, tedy náhodnou volbu více počátečních bodů, tak lze docílit globálního maxima i tam, kde je lokálních extrémů více [4].



***Obr. 3** Blokové schéma Horolezeckého algoritmu*

Výhodou tohoto algoritmu je to, že po celou dobu procesu je zapotřebí pořád stejného množství času na provedení jednoho kroku, a pokud jsme schopni nalézt monotónní heuristickou

funkci, tak dokáže horolezecký algoritmus rychle dosáhnout optima. Nevýhodou je ta skutečnost, že většinou potřebujeme optimalizovat komplexní a složitější problém a v tomto případě dokážeme pomocí tohoto algoritmu dosáhnout jen lokálního extrému.

1.1.1 Ukázka horolezeckého algoritmu

1.1.1.1 Kód v programu MATLAB

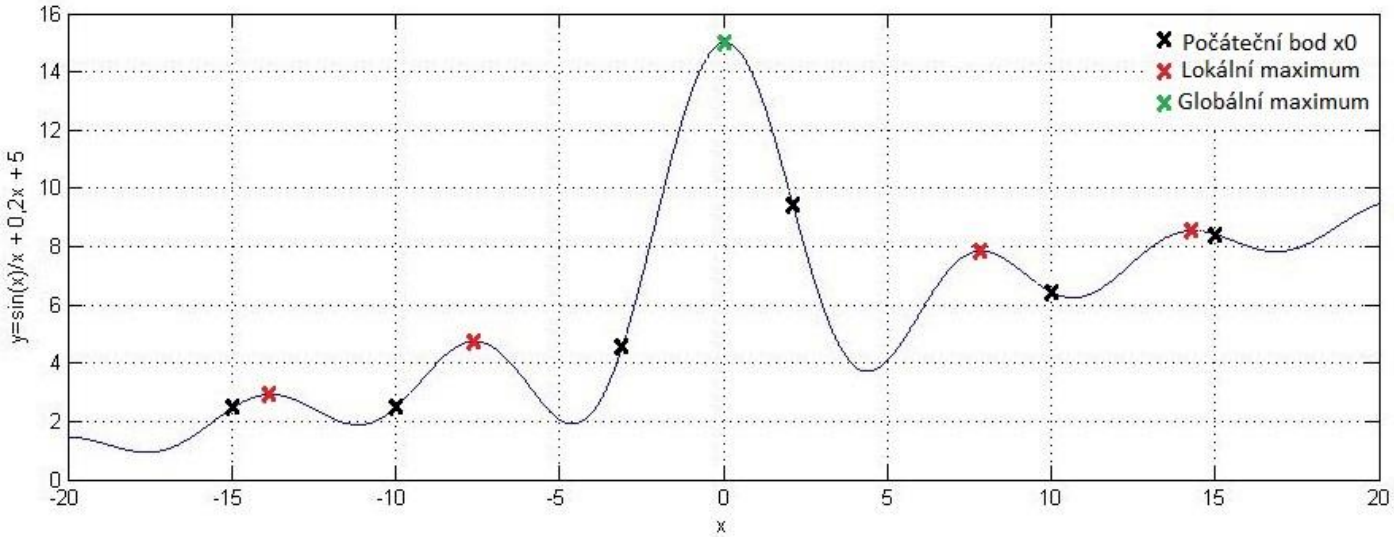
```
x=-20:0.01:20;                                          určení definičního oboru a kroku
y=10*sin(x)./x+0.2*x+5;                                 zadání funkce
x0=-15;                                                 volba počátečního bodu
y0=10*sin(x0)/x0+0.2*x0+5                               hodnota heuristické funkce v bodě x0
y1=10*sin(x0+0.01)/(x0+0.01)+0.2*(x0+0.01)+5;           hodnota heuristické funkce v bodě x0+0.01
y2=10*sin(x0-0.01)/(x0-0.01)+0.2*(x0-0.01)+5;           hodnota heuristické funkce v bodě x0-0.01
n=0;                                                    deklarování proměnné pro počítání kroků
tic                                                     funkce pro odstartování měřiče času
if y1 > y2                                              výběr vyšší hodnoty heuristické funkce
  while y1 > y0
  n=n+1;
  x0=x0+0.01;
  y0=10*sin(x0)/x0+0.2*x0+5;
  y1=10*sin(x0+0.01)/(x0+0.01)+0.2*(x0+0.01)+5;
end                                                     tělo algoritmu
else
  while y2 > y0
  n=n+1;
  x0=x0-0.01;
  y0=10*sin(x0)/x0+0.2*x0+5;
  y2=10*sin(x0-0.01)/(x0-0.01)+0.2*(x0-0.01)+5;
end
end;
toc                                                     funkce pro ukončení měřiče času
 y0=10*sin(x0)/x0+0.2*x0+5                              vyhodnocení heuristické funkce v optimalizovaném bodě
n                                                       vypíše počet kroků
```

1.1.1.2 Výsledky optimalizace

Z **Tab. 1** a **Graf 2** je patrné, že pokud zvolíme vhodně počáteční body x_0, tak pomocí horolezeckého algoritmu nalezneme globální maximum funkce 1.3, jako jsou body x= {-3,3}. V ostatních případech se algoritmus zastavil pouze na lokálních extrémech. Protože v programu MATLAB není optimalizace horolezeckým algoritmem předem vytvořena, tak jsem napsal kód vlastní. Porovnáním jednotlivých počátečních bodů také můžeme vidět, že s rostoucím počtem kroků *n*, roste také čas, po který algoritmus pracuje. I když je počet kroků poměrně vysoký, až 306 kroků, tak čas se nijak výrazně nemění a stále se pohybuje v jednotkách milisekund. Například doba hledání z bodu $x_0 = 15$ je roven $t = 3{,}5$ ms při 64 krocích. Oproti tomu čas, po který běžel algoritmus z bodu $x_0 = -3$ je $t = 7$ ms i přes to, že počet kroků je 306, což je téměř pětinásobek kroků, ale jen dvojnásobek výpočetní doby.

x_0	y_0	$y_{0\_výsledné}$	t	n
-15	2,4335	2,9239	0,0040 s	121
-10	2,4560	4,7541	0,0060 s	243
-3	4,8704	15,0060	0,0070 s	306
3	6,0704	15,0060	0,0065 s	296
10	6,4560	7,8446	0,0055 s	212
15	8,4335	8,5512	0,0035 s	64

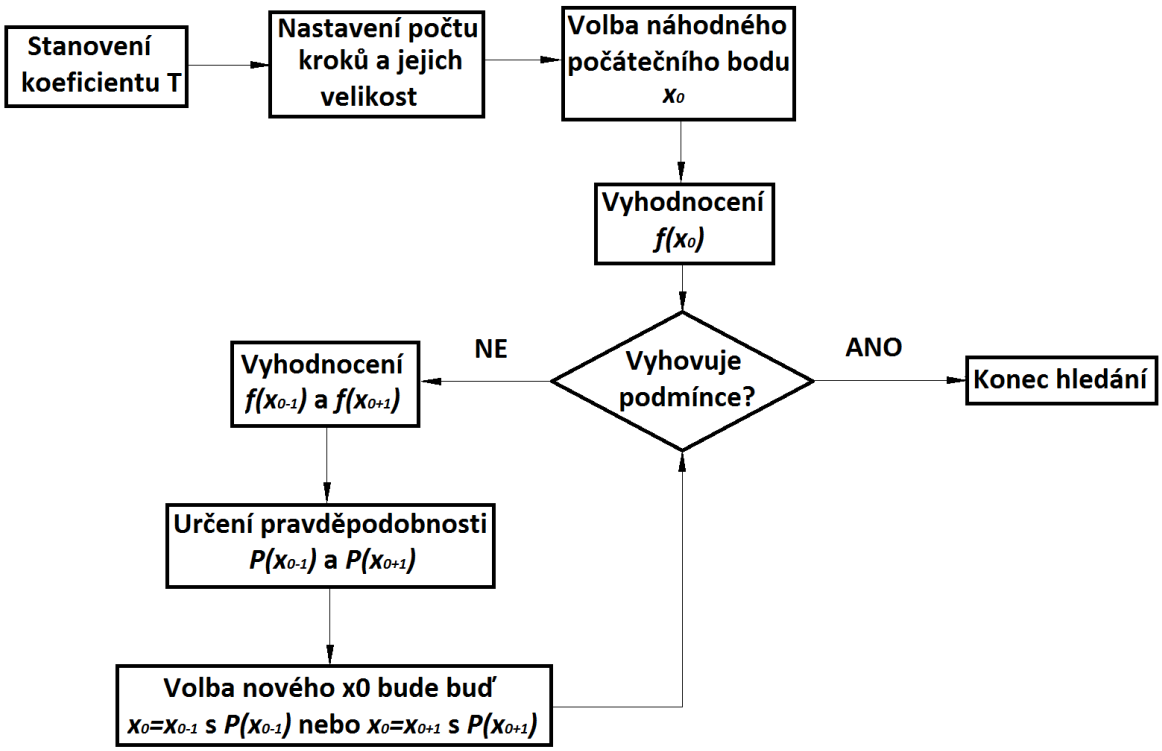
Tab. 1 *Výsledky optimalizace horolezeckým algoritmem*



Graf 2 *Výsledky optimalizace horolezeckým algoritmem*

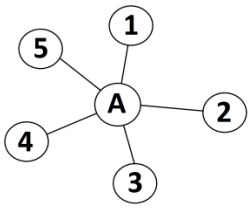
1.2 Simulované žíhání

Je to algoritmus založený na principu takzvaného žíhání kovů, což je proces kontrolovaného ochlazování. Cílem tohoto procesu je docílit kvalitnějšího atomární uspořádání uvnitř kovů. Při chladnutí se snižuje energie jednotlivých atomů, což má za následek zmenšující se kinetickou energii atomů. Atomy s menší kinetickou energií kmitají pomaleji a upevňují tak svoji pozici v krystalické struktuře materiálu (za předpokladu, že daný typ materiálu není amorfní). Čím nižší energetické hladiny atomy dosáhnou, tím kvalitnějším se kov stane. Různé materiály krystalizují v rozdílných krystalických strukturách, ale předpokládejme, že kov, na jehož základě je postavena analogie tohoto algoritmu krystalizuje v blíže nespecifikované struktuře. Atomy v takové struktuře mají nějakou energii, která koresponduje s hodnotami heuristické funkce. Pokud chladnutí probíhá moc rychle, tak se atomy nestihnou správně uspořádat do krystalických struktur, a proto se uměle chladnutí prodlužuje. Tomuto procesu se říká žíhání. Zpomalováním chladnutí vlastně optimalizujeme energii materiálu na minimální hodnotu, ale ne nejrychlejší možnou cestou, jak by tomu bylo u horolezeckého algoritmu, děje se tomu zpomaleně a s dávkou náhodného výběru.



Obr. 4 *Blokové schéma simulovaného žíhání*

Samotný algoritmus simulovaného žíhání je založen na horolezeckém algoritmu a to tak, že ze začátku prohledávání probíhá náhodně. Nejprve je zvoleno několik bodů v okolí výchozího bodu a ty jsou vyhodnoceny. Podle rovnice 1.5 se určí pravděpodobnost, s jakou se přesune stávající bod na novou pozici. S postupem času se ale optimalizační proces stane méně náhodným a to tak, že již zmíněná pravděpodobnost bude méně závislá na výhodnosti dalšího kroku. Jinými slovy přestane pravděpodobnost výrazně ovlivňovat rozhodování a ze simulovaného žíhání se prakticky stane horolezecký algoritmus [5].



Obr. 5 *Prohledávání prostoru z bodu A do nového bodu*

Uvážíme-li situaci, kterou zobrazuje **Obr. 5**, tak z bodu *A* je možné přejít do nových bodů *1 – 5*. Pokud by se jednalo o horolezecký algoritmus, tak se pro každý z nových bodů vyčíslí hodnota heuristické funkce a ten bod, který by nejvíce vyhovoval hledaným požadavkům, se stane novým bodem *A*. Pokud optimalizujeme pomocí algoritmu simulovaného žíhání, tak každý z bodů *1 – 5* má jen určitou pravděpodobnost, že se stane novým bodem *A*. Čím jsou body 1 – 5 vhodnější s tím větší pravděpodobností se stanou novým bodem *A* [5]. V **Tab. 2** je pak znázorněno, jakým způsobem se určí pravděpodobnost nového bodu. Aby se jednalo o algoritmus simulovaného žíhání, tak musíme uvážit vliv náhody, který má analogii v již zmíněné teplotě žíhaného kovu. Čím vyšší je teplota, tím náhodněji se pohybují atomy, což se v algoritmu projeví

tak, že zvolíme-li velkou hodnotu koeficientu reprezentující teplotu, tak následující bod bude vybrán mnohem náhodněji, než by tomu bylo při nízké teplotě. Rovnice popisující tyto skutečnosti má pak následující tvar:

$$P(A,n) = \frac{1}{1 + e^{\frac{-\Delta E}{T}}} \tag{1.5}$$

Kde *P(A,n)* je pravděpodobnost, s jakou se bod *A* přesune do nového bodu $n \in \langle 1,5 \rangle$, T je již zmíněný koeficient reprezentující teplotu kovu a ΔE je rozdíl hodnot heuristických funkcí v bodě *A* a novém bodě *n*. Ne náhodou je tento rozdíl značen právě ΔE, jedná se o analogii rozdílu energií, kterých nabývají atomy v dvou různých stavech [5]:

$$\Delta E = h(n) - h(A) \tag{1.6}$$

h(n)	$-\Delta E$	$e^{-\Delta E/T}$	**P**
80	27	14,88	0,06
100	7	2,01	0,33
107	0	1,0	0,5
120	-13	0,27	0,78
150	-43	0,01	0,99

Tab. 2 *Určení pravděpodobnosti nového bodu A_1, pro $h(A_0)=107$ a T=10 [5]*

T	$e^{-\Delta E/T}$	***P***
1	*0,000002*	*1,0*
5	*0,074*	*0,93*
10	*0,27*	*0,78*
20	*0,52*	*0,66*
50	*0,77*	*0,56*
10^{10}	*0,9999*	*0,5*

Tab. 3 *Vliv teploty na P(A,n) pro h(n)=120 [5]*

Jak je znázorněno v **Tab. 2** a **Tab. 3** tak s rostoucí hodnotou heuristické funkce nového bodu *h(n),* roste také pravděpodobnost, že se bod *A* přemístí do tohoto bodu, a zároveň s rostoucí teplotou roste i náhodnost volby. V opačném případě, kdy je teplota velice nízká, tak se simulované žíhání stává horolezeckým algoritmem. S nahlédnutím na proces optimalizace simulovaným žíháním jako na celek je nejprve prohledávání nahodilé a bod A se přemisťuje náhodně resp. jen s malým přihlédnutím na hodnotu heuristické funkce nového bodu. Po několika předem určených krocích se zmenší teplota a optimalizace se soustřeďuje okolo lokálních optim. V ideálním případě, tedy při vhodném zvolení teplotního poklesu se bod *A* bude nalézat v globálním optimu.

Výhodou simulovaného žíhání je to, že dokáže nalézt globální extrém v prohledávaném prostoru a doba potřebná na jeho prohledání je po celou dobu konstantní. Úspěšnost nalezení globálního optima je ale úměrná počtu kroků a může se tedy stát, že stejně jako u samotného horolezeckého algoritmu bude optimalizovaná hodnota pouze lokálním optimem. Nevýhodou je také složitost volby počáteční teploty a kroku, po kterém tato teplota klesne. Nevhodným zvolením těchto parametrů dochází k prodloužení doby hledání. Může se ale stát i to, že algoritmus vůbec globální optimum nenajde.

1.2.1 Ukázka simulovaného žíhání

1.2.1.1 Kód v programu MATLAB

1. *File > New > Script*

2. *function y= f(x)*
   *y=-10*sin(x)./x-0.2*x-5;*

3. *File > Save*

4. *x0= -15;*
   lb= -20;
   ub= 20;
   [x,fval,exitflag,output] = simulannealbnd(@f,x0,lb,ub)

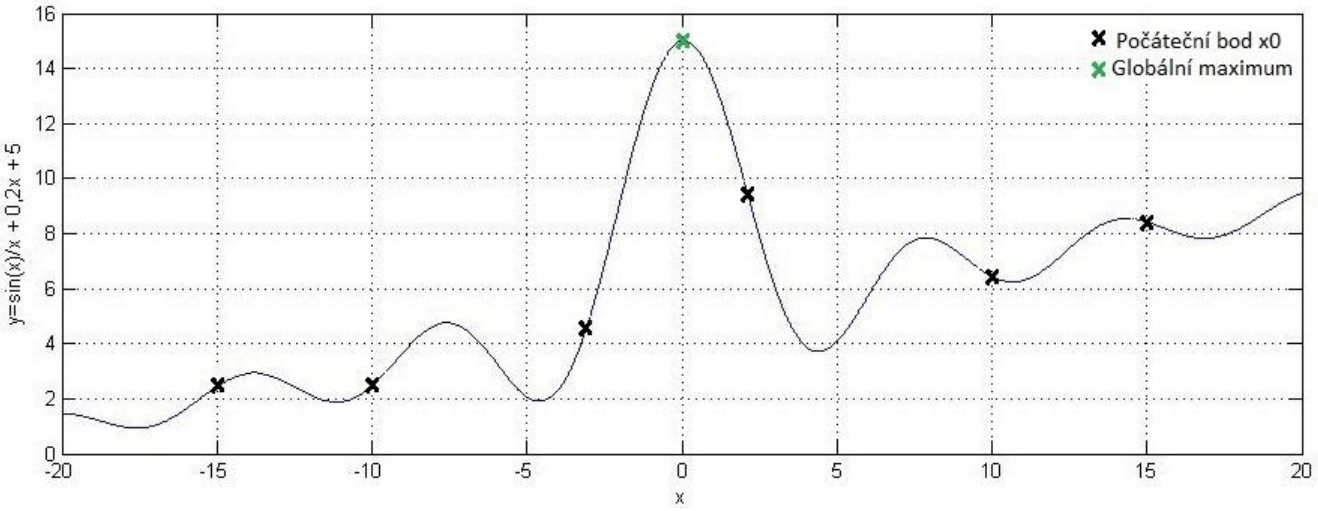
Kde *@f* je funkce dříve definovaná jako y=f(x), x_0 je počáteční bod, *lb* je spodní hranice osy x a *ub* je horní hranice osy x. V kroku 2 je funkce zadána s opačnými znaménky proto, že program MATLAB provádí optimalizaci jako minimalizování. V našem případě hledáme maximum, a proto je nutné upravit znaménkovou konvenci [11].

1.2.1.2 Výsledky optimalizace

Z **Tab. 4** a **Graf 3** je zřejmé, že algoritmus simulovaného žíhání dokáže nalézt globální maximum, a to nezávisle na volbě počátečního bodu x_0. V porovnání s horolezeckým algoritmem, je simulované žíhání úspěšnější, avšak za cenu potřeby většího výpočetního výkonu, což je patrné z **Tab. 1** a **Tab. 4**. Ze srovnání počtu kroků a doby výpočtu oproti horolezeckému algoritmu vyplývá, že algoritmem simulovaného žíhání trvá výpočet téměř desetinásobek času a je zapotřebí přibližně pětinásobného počtu kroků. Dále je možné z **Tab. 4** pozorovat vliv volby počátečního bodu, a to takový, že s rostoucí vzdáleností počátečního bodu od maxima funkce, roste jak počet kroků, tak i potřebný výpočetní čas. I při optimalizování takovéto jednoduché funkce trval výpočet několikanásobně déle, což by se při řešení rozsáhlejšího problému, jako například elektrického motoru, značně projevilo. Zejména uvážíme-li, že pomocí algoritmu bude nutné řešit funkce více proměnných, kde každá proměnná bude výrazně složitější než takto jednoduchá funkce. V tomto případě MATLAB obsahoval přednastavenou optimalizaci simulovaným žíháním, a proto jsem využil této možnosti [5].

x_0	y_0	$y_{0\_výsledné}$	t	n
-15	2,4335	15,0060	0,4063	1462
-10	2,4560	15,0060	0,3906	1310
-3	4,8704	15,0060	0,2031	679
3	6,0704	15,0060	0,1875	663
10	6,4560	15,0060	0,2969	993
15	8,4335	15,0060	0,4063	1430

Tab. 4 *Výsledky optimalizace algoritmem simulovaného žíháním*



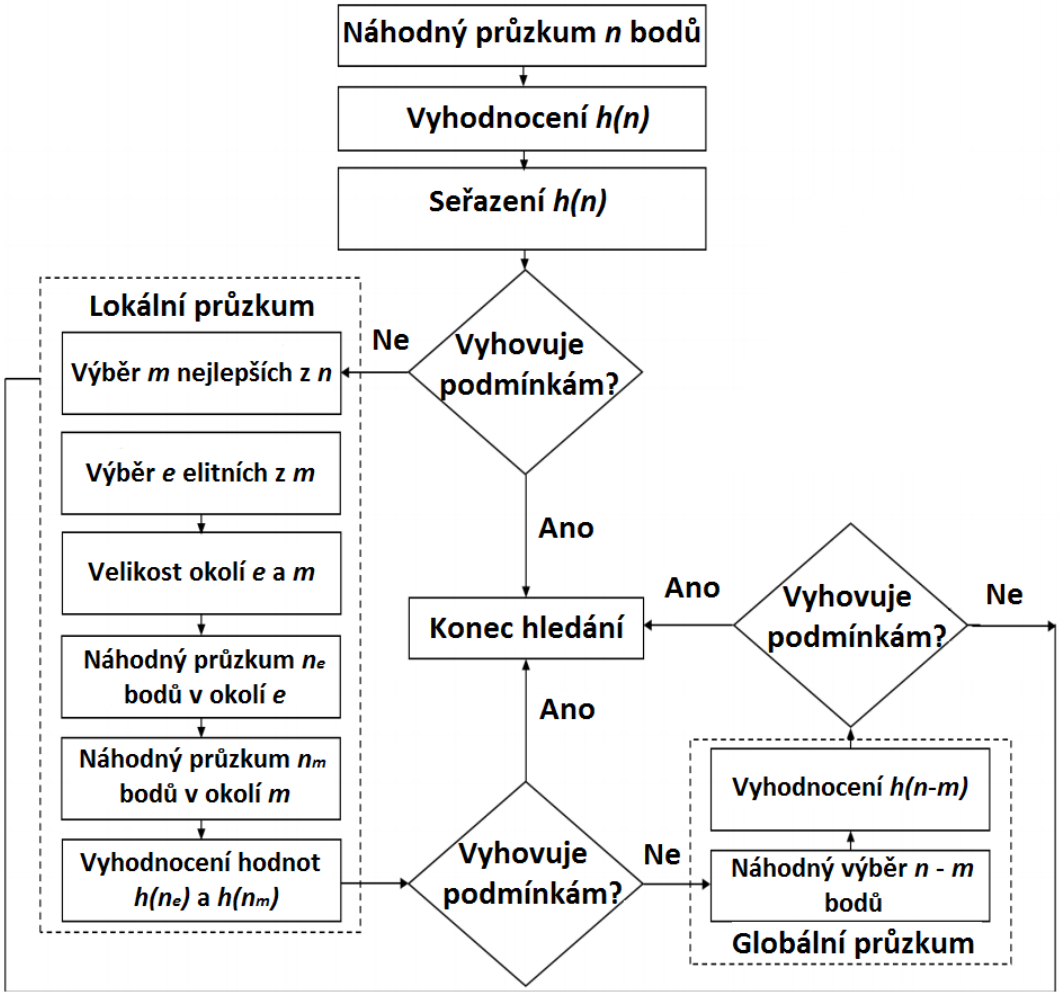
Graf 3 *Výsledky optimalizace simulovaným žíháním*

1.3 Včelí algoritmus

Řadí se do skupiny evolučních algoritmů a ještě přesněji do skupiny algoritmů, které se označují jako SOAs (Swarm Optimization Algorithms). Princip těchto metod, tedy i včelího algoritmu je označován jako populační, což znamená, že pracuje současně s celou skupinou jedinců. Na rozdíl od horolezeckého algoritmu nebo simulovaného žíhání, kde se optimum hledá krok po kroku, tak populační algoritmy pracují s řadou kroků zároveň, tedy s populací. Touto populací mohou být například včely, ale také mravenci, ptáci, ryby, nebo dokonce i buňky v lidském těle [7]. Všechny tyto algoritmy se velice hodí pro řešení problému, jejichž průběh je členitý, nebo se v řešení problému vyskytuje více lokálních extrémů a to z toho důvodu, že prohledávají celý prostor současně.

Včelí algoritmus je postaven na základě chování včelího roje, kde jednotlivci z populace náhodně prohledávají žádaný prostor a sbírají informace, resp. vyhodnocují hodnotu fitness funkce a místo, ve kterém se tato hodnota nachází. Toto hledání je obdoba hledání potravy. Včely se rozletí do okolí v určitém počtu, například 50 nebo 100 popřípadě více jednotlivců, do náhodných směrů. V přírodě je prohledávaný prostor trojrozměrný, ovšem algoritmus dokáže fungovat i v prostorech s jiným počtem dimenzí jako dvojrozměrném, nebo čtyřrozměrném atd.

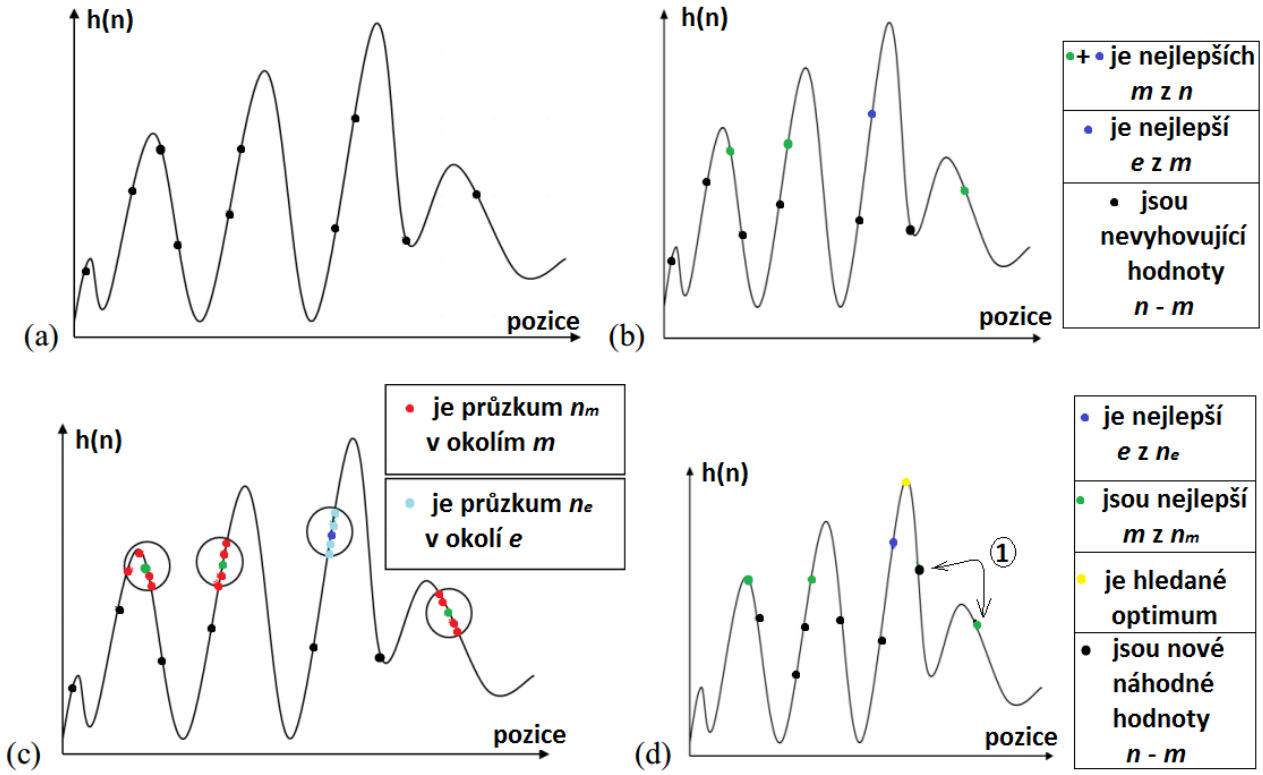
Když včely naleznou nějaký zdroj potravy, tedy květ, tak vyhodnotí vzdálenost a směr od úlu a velikost zdroje potravy. V prostředí algoritmu to znamená vyhodnocení fitness funkce v bodě o daných souřadnicích. S touto informací se včely vrátí do úlu a tam si navzájem předají informace o zdroji potravy, který nalezly. Na základě těchto informací se dále rozhodnou, jakým zdrojům potravy dají přednost a které vynechají z dalšího hledání. Poté se určitý počet včel, dejme tomu 30%, rozlétne podrobněji prozkoumat vytipované oblasti a zbytek dál náhodně prozkoumává všechny směry. To, jestli se včely rozdělí na 30% a 70% nebo nějak jinak, není nijak striktně dáno. U algoritmu je to parametr, který si musíme zvolit a jeho změnou jsme pak schopni měnit průběh algoritmu. Po druhém prohledávání, se opět předají informace a porovnají, jestli předchozí kandidáti zdrojů potravy jsou stále nejlepší volbou, nebo se během náhodného prohledávání neobjevilo nějaké nové místo. Pokaždé když se včely rozhodují, tak jednají na základě porovnání s uspokojivou mírou potravy, což pro včelí algoritmus znamená, že jednotlivé hodnoty fitness funkce se porovnávají s nějakou optimální hodnotou nebo se kontroluje, jestli mezi jednotlivými prohledáváními není již dostatečně malý rozdíl [7].



Obr. 6 *Blokové schéma včelího algoritmu [7]*

Při optimalizaci za pomocí včelího algoritmu se nejprve náhodně zvolí *n* pozic v prohledávaném prostoru a vyhodnotí se hodnoty fitness funkce *h(n)* v těchto pozicích viz **Obr. 7a**. Hodnoty *h(n)* se posléze porovnají s požadovanou hodnotou (optimem) a seřadí se v pořadí od nejvíce vyhovující po tu nejméně vyhovující. Pokud některá z hodnot *h(n)* vyhovuje, tak je optimalizace u konce, v opačném případě se provede krok, ve kterém se prohledává okolí *m*

nejlepších bodů viz **Obr. 7b**. Velikost prohledávaného okolí a počet kroků n_m, resp. n_e je nutné vhodně zvolit. Pokud bude prohledávaná oblast příliš velká, tak bude algoritmus příliš dlouho setrvávat v jednom kroku a může se výrazně prodloužit počet kroků potřebných k nalezení maxima. V opačném případě bude algoritmus zbytečně prohledávat celý prostor a nesoustředí potřebnou výpočetní sílu na preferované body. V dalším kroku se opět vyhodnotí $h(n_m)$, resp. $h(n_e)$ a seřadí se. Zde opět proběhne kontrola, zda některá z hodnot $h(n_m)$, resp. $h(n_e)$ nevyhovují hledanému optimu viz **Obr. 7c**. Není-li optima dosaženo, tak přijde na řadu nové náhodné prohledávání celého prostoru, ale tentokrát je prohledáno $n - m$ kroků. Všechny hodnoty $h(n)$ včetně těch nejvhodnějších $h(n_m)$, resp. $h(n_e)$ se znovu seřadí, viz **Obr. 7d**. V tomto bodě se algoritmus vrací zpět na začátek a opakuje se, dokud nesplní některé ze zadaných požadavků. Podrobné blokové schéma je znázorněno na **Obr. 6** [7].



***Obr. 7** Grafické znázornění průběhu včelího algoritmu [7]*

Výhodou včelího algoritmu je schopnost podrobného nalezení jak lokálních, tak i globálních extrémů. Je jednoduchý na použití a snadno kombinovatelný s jinými algoritmy popřípadě jej lze hybridně kombinovat. Nevýhodou je pak nutnost manuálního nastavení řady parametrů a jejich postupné ladění. Při nevhodné volbě těchto parametrů se extrémně prodlužuje čas konvergence. Na druhou stranu lze při vhodném zvolení parametrů docílit menšího počtu kroků i kratší doby výpočtu. Ve všech případech hraje velkou roli náhodné prohledávání. Při stejné volbě parametrů se snadno může stát, že počet kroků bude u prohledávání stejného prostoru pokaždé různý [7].

1.3.1 Ukázka algoritmu včelích kolonií

1.3.1.1 Kód v programu MATLAB

```
x=-20:0.01:20;
y=10*sin(x)./x+0.2*x+5;
P=10;                                   počet náhodně prohledávaných bodů
m=6;                                    výběr m nejlepších
ne=20;                                  počet prohledávaných bodů v okolí jednoho nejlepšího z m
nm=10;                                  počet prohledávaných bodů v okolí ostatních bodů m
y00=0;                                  počáteční hodnota y nejlepšího bodu
x00=0;                                  počáteční hodnota x nejlepšího bodu
z=0;                                    počáteční hodnota náhodně generovaného čísla
A= [1:P];
B= [1:P];                               deklarování matic
C= [1:m];
D= [1:m];
E= [1:m];
Best=0.0002;                            deklarování proměnných potřebných k provedení smyčky
loop=0.0001;                            respektive podmínky while
while loop~=0                           smyčka, která se opakuje tak dlouho, až 3 po sobě jdoucí
                                        cykly nenajdou lepší hodnotu
 loop=0;
  for k=1:4
    for a= 1:P
    z=round(rand(1)*40-20);             generování náhodných celých čísel v rozsahu -20 až 20 do
      if z==0                           matice A
      z=z+0.01;
     end
  A(1,a)=z
  end

for a= 1:m
  C(1,a)=0;
  D(1,a)=0;                             nulování matic
  E(1,a)=0;
end

for a= 1:P
  x0= A(1,a);                           výpočet y hodnot pro hodnoty z matice A
  y=10*sin(x0)./x0+0.2*x0+5;
  B(1,a)=y;

  if y > y00
    x00=x0;
    y00=y;                              nulování souřadnic nejlepšího bodu aktuální smyčky
  end
end

for a= 1:P
  u=A(1,a);
  v=B(1,a);                             prohledávání matice B s náhodnými hodnotami y a vybrání
                                        m nejlepších hodnot. Hodnoty y jsou uloženy do matice D
  if v > D(1,m)                         a hodnoty x do matice C
    C(1,m)=u;
    D(1,m)=v;
  end

for b= 1: (m-1)
  if D(1,m+1-b) > D(1,m-b)
  V=D(1,m-b);
```

```
 D(1,m-b)= D(1,m+1-b);
 D(1,m+1-b)=V;                                   setřídění nejlepších m hodnot v matici D
 U=C(1,m-b);
 C(1,m-b)= C(1,m+1-b);
 C(1,m+1-b)=U;
 end
end
end

x00=x00-0.01*ne/2;
for a= 1:ne
y=10*sin(x00+a*0.01)./(x00+a*0.01)+0.2*(x00+a*0.01)+
5;
 if y > y00                                      prohledání okolí nejlepšího bodu
 y00=y;
 end
end

D(1,1)=y00;

for a= 1:(m-1)
 for b= 1:nm
 y=10*sin(C(1,a+1)-0.01*nm/2+b*0.01)./ (C(1,a+1)-
0.01*nm/2+b*0.01)+0.2*(C(1,a+1)-
0.01*nm/2+b*0.01)+5;                             prohledání okolí ostatních m bodů
   if y > D(1,a+1)
   D(1,a+1)=y;
   end
 end
end

for a= 1: (m-1)
 if D(1,m+1-a) > D(1,m-a)
 V=D(1,m-a);
 D(1,m-a)= D(1,m+1-a);
 D(1,m+1-a)=V;                                   opětovné setřídění po prohledání okolí
 U=C(1,m-a);
 C(1,m-a)= C(1,m+1-a);
 C(1,m+1-a)=U;
 end
end

for a= 1:m
 if D(1,a) > E(1,a)
 E(1,a)=D(1,a);
 end                                             uložení nových nejlepších hodnot do matice E
end
Best= E(1,1);
loop=loop+Best;

 end
loop=4*Best-loop;                                ukončení smyčky
end
```

1.3.1.2 Vyhodnocení

Algoritmus dokázal pokaždé nalézt maximální hodnotu y=15,006. Tuto hodnotu dokázal oproti simulovanému žíhání nalézt velice rychle. Při nastavování jsem zjistil, že volba parametru *P*, *ne* a *nm* je podstatná pro průběh algoritmu, a to tak, že při velkém *P* je více pravděpodobné, že už při náhodné volbě hodnot x se algoritmus trefí do blízkosti maximální hodnoty a tím pádem se značně omezí počet smyček. V opačném případě je sice počet smyček cyklu velký, ale počet kroků na smyčku je menší. V kombinaci s volbou počtu prohledávaných bodů v okolí m nejlepších bodů lze algoritmus nastavovat dle potřeby. I za předpokladu, že se nepodařilo algoritmus optimálně nastavit, tak pro výpočet bylo zapotřebí 12 kroků a výpočet trval 0,0224 s, což je přibližně desetinásobně kratší doba, než u simulovaného žíhání.

U tohoto algoritmu nemá z principu smysl uvažovat počáteční body x_0 z rovnice 1.4, protože algoritmus počáteční body volí náhodně. V porovnání s horolezeckým algoritmem, nebo simulovaným žíháním, je algoritmus včelích kolonií mnohem vhodnější k průzkumu členitějších funkcí. Na základě zkušeností s volbou parametrů lze očekávat, že s rostoucí složitostí funkce se bude zvětšovat výpočetní náročnost algoritmu. V případě použití algoritmu pro výpočet elektromotoru by nejspíše optimalizování netrvalo řádově milisekundy, ale spíše jednotky až desítky minut. Vzhledem k tomu, že při takovémto, nebo podobném výpočtu bude za potřebí algoritmus postupně ladit, tak výpočetní doba hraje významnou roli.

1.4 Mravenčí kolonie

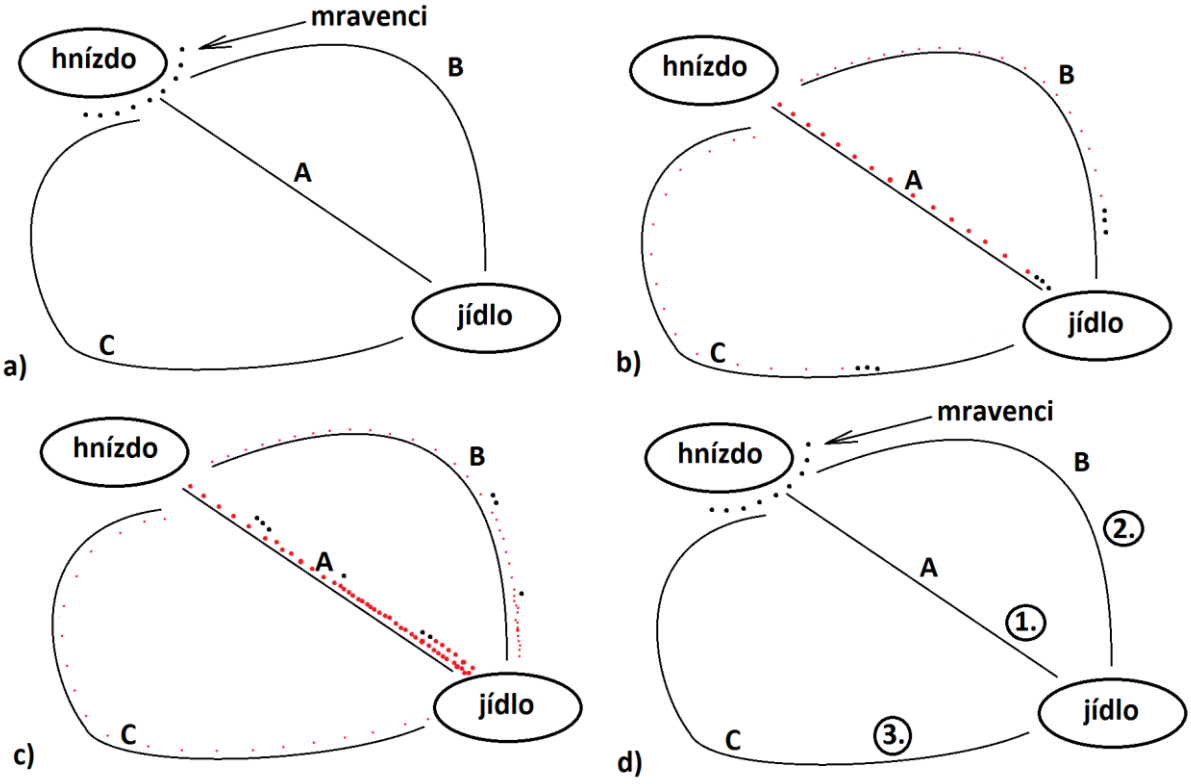
Podobně jako u včelího algoritmu se i v tomto případě jedná o algoritmus, který se inspiruje v chování živočichů, tedy v tomto případě mravenců. Jak včely, tak i mravence lze zařadit do kategorie takzvaného sociálního hmyzu, což znamená, že pro jednotlivce je důležitější přežití kolonie než vlastní zištnost. Algoritmus je postaven na principu hledání nejkratší cesty ke zdroji potravy a na množství potravy nalézající se na konci této cesty. Při chůzi totiž mravenci vylučují feromon, který po určitou dobu zanechává stopu pro ostatní mravence. Ti se řídí množstvím feromonu, který ucítí, a to tak, že čím je více feromonu v daném místě, tím je větší šance, že se pro tuto cestu mravenci rozhodnou. Pokud mravenci prohledávají veliký prostor, s řadou velkých zdrojů potravy, tak se může stát, že se všichni mravenci nesoustředí pouze na jeden nejvýhodnější zdroj potravy. S postupným působením feromonu se mravenci soustředí do několika nejvýhodnějších oblastí. Tyto oblasti si v matematické analogii můžeme představit jako lokální extrémy například funkce 1.3. Jednoduchá ukázka hledání nejkratší cesty k jednomu zdroji potravy je zobrazena na **Obr. 8**.

Jak je zobrazeno na **Obr. 8a**, tak na začátku je dané náhodně, kterou cestou se mravenci vydají, resp. na každou cestu se vydají se stejnou pravděpodobností. Při chůzi produkují určité množství feromonu. Skuteční mravenci produkují feromon průběžně, viz **Obr. 8b**, ale pro zjednodušení algoritmu uvažujeme, že množství feromonu je vyloučeno až na konci cesty. Toto množství feromonu je:

$$\tau(x+1) = \Delta\tau_{h,j\ best} + (1-q)\tau_{h,j}(x) \tag{1.7}$$

kde q je míra vypařování feromonu, koeficienty h,j značí, že se jedná o cestu mezi hnízdem a jídlem a $\Delta\tau_{h,j\ best}$ je parametr hodnotící předchozí množství feromonu, se kterým se mravenec setkal, resp. to nejlepší z nich. Z této rovnice tedy plyne, že chování mravenců je silně závislé na množství feromonu, ale stejně tak důležité je i vyhodnocení tohoto množství. Je také zřejmé, že záleží na zkušenosti každého mravence. Pokud se například v určité blízké oblasti nachází velké

množství potravy, tak na cestě vedoucí k tomuto zdroji potravy bude značné množství feromonu. Ovšem mravenci, kteří ze začátku prohledávali opačnou stranu prostoru, můžou vlivem odpařování feromonu zůstat izolovaní v lokálním extrému blízkému jejich začáteční pozice.



Obr. 8 *Analogie pro algoritmus mravenčích kolonií*

Vrátíme-li se k parametru $\Delta\tau_{h,j\,best}$ z rovnice 1.7, tak pro první cestu je tento parametr roven 0, protože mravenci nemají zatím žádnou předešlou zkušenost. Pro ostatní cesty je pak tento parametr roven převrácené hodnotě délky cesty:

$$\Delta\tau_{h,j}^{best} = \frac{1}{h(x)_{best}} \tag{1.8}$$

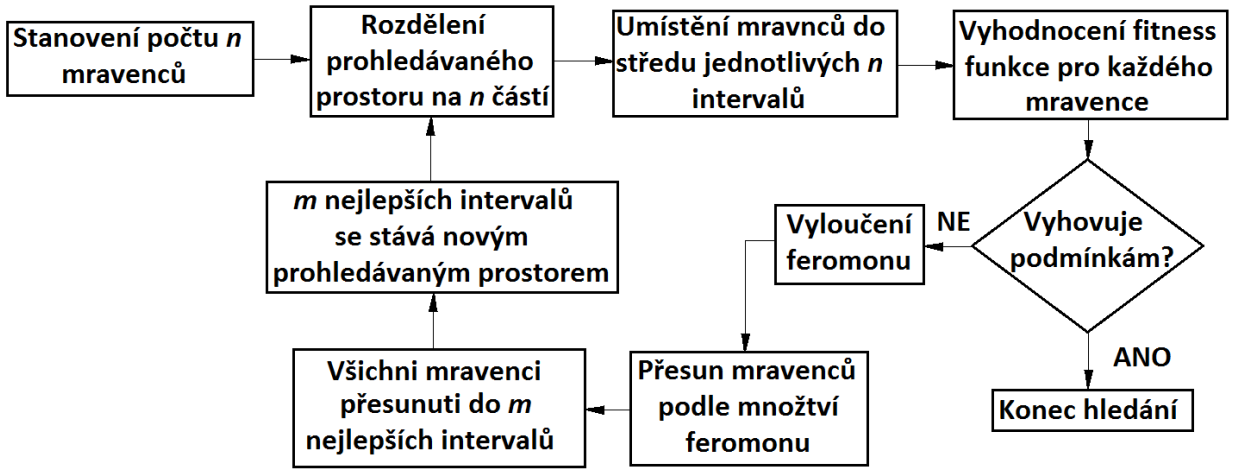
pro každou další cestu pak mravenci vyhodnocují výhodnost cesty podle množství feromonu příslušící této cestě. To, po které cestě se vydají, je pak pro potřeby algoritmu určeno následovně:

$$P_{h,j} = \frac{[\tau_{h,j}(x)]^{\alpha}[f(x)_{h,j}]^{\beta}}{\sum_{k=A,B,C}[\tau_{h,j,k}(x)]^{\alpha}[f(x)_{h,j,k}]^{\beta}} \tag{1.9}$$

pro první případ je pravděpodobnost pro každou z cest stejná, resp. $P_{h,j} = \frac{1}{n}$, kde n je celkový počet mravenců. Koeficienty α a β jsou parametry, které je nutné vhodným způsobem nastavit vzhledem k charakteru fitness funkce *f(x)*.

Jak je vidět na **Obr. 8c**, tak na kratší cestě se vyskytuje více feromonu, protože tudy první mravenci prošli dříve a tím pádem dříve uvolnili feromon, což má vliv na rozhodování dalších mravenců. Mravenci co dorazí k jídlu později, tedy šli delší cestou, vycházejí ze zkušenosti prvních mravenců. Postupem času se také projeví vypařování feromonu, které má za následek

snižování pravděpodobnosti výběru méně frekventovaných, tedy kratších cest. Pravděpodobnost výběru nejkratší cesty se tedy zvětšuje, až nakonec všichni mravenci chodí kratší cestou.



***Obr. 9** Blokové schéma mravenčího algoritmu*

Výhodou tohoto algoritmu je, že dokáže nalézt optimální řešení, v tomto případě, tedy nejkratší cestu, resp. globální minimum. Pro některé typy praktických úloh se charakter tohoto algoritmu hodí, ale je obtížné nastavit parametry a heuristickou funkci tak, aby nedocházelo ke konvergenci k lokálnímu optimu.

1.4.1 Ukázka mravenčího algoritmu

1.4.1.1 Kód v programu MATLAB

```
x=-20:0.01:20;
y=10*sin(x)./x+0.2*x+5;
n=40;                                   počet mravenců
nn=0;
m=0;                                    deklarování proměnných potřebných pro ukládání
loop=1;                                 mezivýpočtů
loop2=1;
Pom=[1,1];                              pomocná matice
A=[1,n];                                matice se začátky intervalů
B=[1,n];                                matice s konci intervalů
C=[1,n];                                matice se středy intervalů, ve kterých jsou mravenci
D=[1,n];                                matice s hodnotami fitness funkce pro každého mravence
M=[1,n];                                matice s rozmístěním mravenců
A(1,1)=-20;                             nastavení začátku prohledávaného prostoru
B(1,n)=20;                              nastavení konce prohledávaného prostoru
Pom(1,1)=0;                             vynulování pomocné matice pro potřeby podmínek alg.
l=(B(1,n)-(A(1,1)))/n;                  určení velikosti intervalů

tic                                     měření doby výpočtu
for c=1:4                               cyklus s podmínkou pro ukončení hledání
 for a=1:n
  A(1,a)= A(1,1) + (a-1)*l;
  B(1,a)= A(1,1) + a*l;
  C(1,a)= A(1,1) + (a-0.5)*l;           naplnění matic A,B,C,D hodnotami podle velikosti intevalu
  D(1,a)= 10*sin(C(1,a))./C(1,a) + 0.2*C(1,a) + 5;   a prvotní rozmístění mravenců do jednotlivých intervalů
  M(1,a)=1;
 end
```

```
loop2=1;
while loop2~=0

Pom=[1,n-m];
mm=0;
k=0;
for a=1:n
 mm=mm+1;
 if M(1,a)~=0
  k=k+1;
  Pom(1,k)=mm;
 end
end

if k>1
 if D(1,Pom(1,2))> D(1,Pom(1,1))
  if M(1, Pom(1,1))>0
   M(1,Pom(1,2))= M(1,Pom(1,2)) + M(1,Pom(1,1));
   M(1,Pom(1,1))= 0;
  end
 end

 if D(1,Pom(1,k-1))> D(1,Pom(1,k))
  if M(1, Pom(1,k-1))>0
   M(1,Pom(1,k-1))= M(1,Pom(1,k-1)) + M(1,Pom(1,k));
   M(1,Pom(1,k))= 0;
  end
 end
end

for a=1:n
 if M(1,a) == n;
  loop=0;
  loop2=0;
 else
  loop=1;
 end
end

while loop~=0

loop=0;
for b=1:3
 nn=nn+1;
 for a=2:(k-1)

  if D(1,Pom(1,a-1)) > D(1,Pom(1,a))
   if M(1,Pom(1,a)) > 0;
    M(1,Pom(1,a-1))= M(1,Pom(1,a-1)) + M(1,Pom(1,a));
    M(1,Pom(1,a))= 0;
   end
  end

  if D(1,Pom(1,a+1)) > D(1,Pom(1,a))
   if M(1,Pom(1,a)) > 0;
    M(1,Pom(1,a+1))= M(1,Pom(1,a+1)) +
M(1,Pom(1,a));
    M(1,Pom(1,a))= 0;
   end
  end
 end
```

nastavení proměnné pro první kolo iterace
začátek smyčky, ve které se prohledávají nejlepší z intervalů

uložení nejlepších hodnot do pomocné matice (v prvním kole iterace jsou v pomocné matici všechny intervaly, v dalších kolech už je to pouze výběr)

prohledání levého krajního intervalu

prohledání pravého krajního intervalů

vyhodnocení podmínek pro další postup algoritmu

začátek smyčky ve které se prohledávají sousedi každého intervalu

cyklus s podmínkou pro ukončení hledání
počítání počtu přesunutí mravenců
prohledání všech intervalů mimo krajních dvou

prohledání sousedů z levé strany (v případě 2D grafu se algoritmus dívá jen doleva a doprava, u vícerozměrného grafu by stran bylo více)

prohledání sousedů z pravé strany

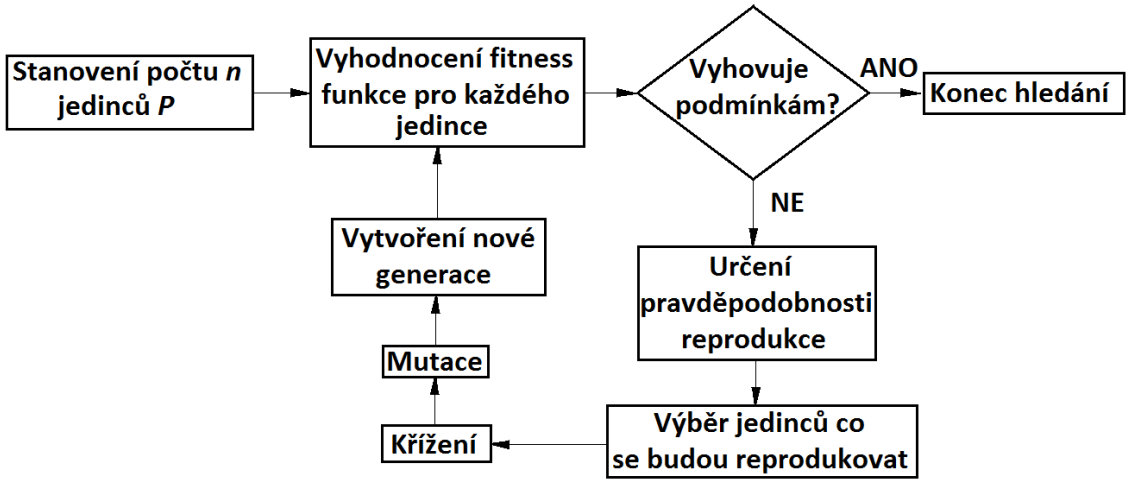
```
 m=0;
 for a=1:n
  if M(1,a)==0
    m=m+1;
  end                                   podmínka a nastavení proměnných potřebných pro ukon-
 end                                    čení smyč jednoho iteračního cyklu
 loop=loop+m;
end
loop=loop - 3*m;
end
end

loop2=1;
m=0;                                    nastavení nového prohledávaného prostoru a velikosti
A(1,1)=A(1,Pom(1,1));                   intervalů
B(1,n)= B(1,Pom(1,1));
l=(B(1,n)-(A(1,1)))/n;
end
clc                                     zobrazení času hledání
toc                                     zobrazení počtu kroků
nn
```

1.4.1.2 Vyhodnocení

Ve všech případech algoritmus nalezl maximální hodnotu. Pro jednoduchost jsem ve všech krocích uvažoval hodnotu feromonu odpovídající hodnosti fitness funkce. Dále jsem předpokládal, že mravenci vždy najdou nejlepší z porovnávaných hodnot (neuvažoval jsem pravděpodobnost ze vzorce 1.9). Tyto zjednodušení jsem přijal pouze proto, že se jedná o ukázku fungování algoritmu, nikoli o podrobný rozbor této metody, který není cílem této práce. Takto nastavený algoritmus prohledával prostor 0,0174s a potřeboval 3 iterační kroky s celkovým počtem 123 přesunutí mravenců. Stejně jako u včelího algoritmu i zde má extrémní vliv počet mravenců v kombinaci s tvarem prohledávaného prostoru. To, jestli zvolit menší počet mravenců nebo větší, nelze snadno odhadnout a je nutné to vyzkoušet. U jednoduché funkce jako je ta mnou použitá je vyřešení otázkou několika minut, ale v případě složitějšího elektromagnetického problému by byl čas mnohonásobně delší. U složitějšího problému by navíc nebylo možné zavádět takové zjednodušující předpoklady, jaké jsem uvažoval já.

1.5 Genetický algoritmus

Základ tohoto algoritmu je položen v samotném základu přírody. Příroda sama provádí jistou formu optimalizace, kterou je jednoduše evoluce. U evoluce se předpokládá, že vzájemným křížením jedinců vzniká nová generace s novými geny, které buď vyhovují, nebo nevyhovují podmínkám pro přežití. Křížení jedinců a kombinování genů tedy vede k tvorbě nové generace s novou kombinací genů. Tato nová generace k přežití využívá informace, kterými disponuje díky zděděným genům. Ti, jejichž geny vyhovují více, mají větší šanci na přežití oproti těm, kteří mají geny horší. To, jakým způsobem přecházejí geny z rodičů na potomky, je zásadní pro evoluci daného druhu. Pokud se genofond celého druhu nepřizpůsobí, tak je pravděpodobné, že nepřežije. Vedle soupeření o přežití mezi jednotlivými druhy existuje také soupeření v rámci jednoho druhu. Soupeří například samci o samice, nebo různé smečky o přísun potravy atd. Výsledkem je již zmíněné optimalizování genofondu druhu, a to tak, že jedinci s výhodnějšími geny přežijí. Především jedinci s vhodnými geny mají možnost je předat dál, to ale neznamená, že vždy jen ti nejlepší předají své geny [8].



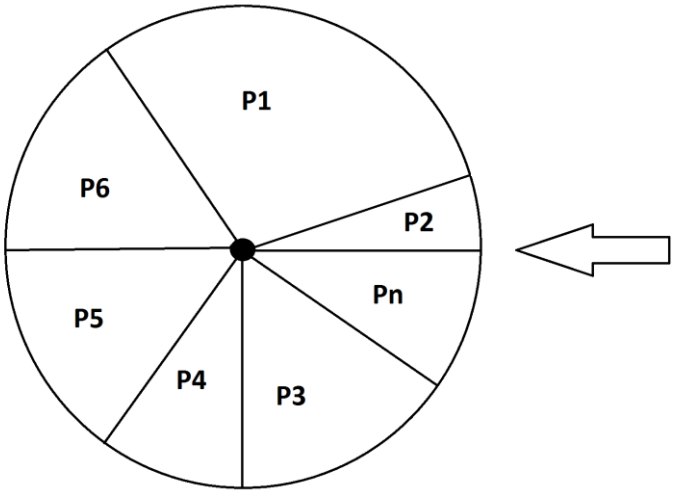
Obr. 10 *Blokové schéma GA*

Podrobným zkoumáním procesu evoluce a pozorováním přírody bylo možné sestavit algoritmus sestávající z několika kroků, který popisuje, jak taková přírodní optimalizace probíhá. V procesu algoritmu je na prvním místě dekódování genu na jednotlivé chromozomy pro každého jedince, resp. určení jednotlivých prvků, ze kterých je složená fitness funkce. Tyto prvky se v genetických algoritmech nazývají vlákna. Jako vlákna si lze představit soubor veličin, které jsou dále tvořeny jinými veličinami respektive chromozomy. Příkladem může být taková fitness funkce, která vyhodnocuje odpor elektrického obvodu (vlákno), který je dále určen součinem proudu a napětí (chromozomy). Jednotlivá vlákna fitness funkce udávají vhodnost genu jako jednu konkrétní hodnotu na základě předem dané podmínky (chceme například, aby odpor obvodu byl minimální, nebo maximální) [8]. Fitness funkce by se pak dala zapsat například v takovémto tvaru:

$$f(n) = \sum_{i=1}^{n_i} P_{i,n} \tag{1.10}$$

Kde *f(n)* je hodnota fitness funkce v bodě *n* a $P_{i,n}$ jsou jednotlivé veličiny v bodě *n*. V každém bodě *n* tedy existuje jedinec (veličina) P_n, nebo skupina jedinců $P_{i,n}$ která může být složena z několika dalších prvků(chromozomů). Algoritmus nejdříve volí několik náhodných jedinců, kterým se říká generace a každému jedinci z generace přiřadí pravděpodobnost, s jakou se bude či nebude reprodukovat. Tato pravděpodobnost se určí na základě hodnoty fitness funkce. Příklad pravděpodobnosti k reprodukci je zobrazen na **Obr. 11**.

V každém vlákně může být například n_i členů a tyto členy mají různé hodnoty (chromozomy) pro různé body *n* v hledaném prostoru, jak je možno vidět v rovnici 1.10. Máme tedy populaci jedinců, jejichž kvality lze vyhodnotit jako hodnotu *h(n)*. Čím více bude vyhovovat hodnota *h(n)* optimu, tím větší šanci k reprodukci jedinec dostane, viz **Obr. 11** [8].



Obr. 11 *Příklad určení pravděpodobnosti reprodukce [8]*

Příklad takovéto reprodukce je znázorněn v **Tab. 5**. V této tabulce jsou naznačeny všechny kroky, tedy výběr jedinců na základě pravděpodobnosti, křížení, a mutace. Tyto kroky jsou postupně vysvětleny níže. Žádný z těchto kroků není určen na základě konkrétních hodnot, jedná se pouze o demonstraci postupu.

Po určení pravděpodobnosti k reprodukci je dalším krokem algoritmu vytvoření nové generace *n* jedinců. Ta je vytvořena náhodně, resp. podle pravděpodobnosti k reprodukci. Tento proces si lze představit, jako bychom otočili kolem z **Obr. 11** n-krát, a na kterého jedince P1 až Pn ukáže šipka, ten se stane reprodukujícím se jedincem P1'[8].

Původní	Reprodukce	chromozomy		Křížení	Příklad nové	
P1	P1'	$P1_1$', $P1_2$', $P1_3$' ,$P1_4$'		P1' x P5'	$P1_1$', $P1_2$', **$P2_3$**' $P5_4$'	P1''
P2	P1‘	$P1_1$', $P1_2$', $P1_3$', $P1_4$'			$P5_1$', $P5_2$', $P1_3$' $P1_4$'	P2‘‘
P3	P5‘	$P5_1$', $P5_2$', $P5_3$', $P5_4$'	➡	P1‘ x P4‘	$P1_1$',$P4_2$', $P4_3$' $P4_4$'	P3‘‘
P4	P4‘	$P4_1$', $P4_2$', $P4_3$',$P4_4$'			$P4_1$', $P1_2$', $P1_3$' $P1_4$'	P4‘‘
P5	P3‘	$P3_1$', $P3_2$', $P3_3$',$P3_4$'		P3‘ x P6‘	$P3_1$', $P3_2$', $P3_3$', $P6_4$'	P5‘‘
P6	P6'	$P6_1$', $P6_2$', $P6_3$',$P6_4$'			$P6_1$', $P6_2$', $P6_3$', $P3_4$'	P6‘‘

Tab. 5 *Příklad pravděpodobnosti reprodukce a křížení nové generace [8]*

V **Tab. 5** je příklad reprodukce, křížení jedinců a vytvoření nové generace. Odtud a z **Obr. 11** je zřejmé, že reprodukce mezi úspěšnějšími jedinci, respektive způsob, jakým se volí reprodukující se dvojice je náhodný (zde může být algoritmus upraven a lze vybírat podle určitých kritérií), stejně tak v kombinování jednotlivých chromozomů. Výsledkem reprodukce je pak nová generace s novými hodnotami počítanými pomocí rovnice 1.10. Pokud některá z nových hodnot *h(n)* vyhovuje hledanému optimu, pak algoritmus končí, v opačném případě následuje další kolo reprodukce, křížení a sestavení nové generace [8].

Problém nastává tehdy, je-li součástí hledaného optimálního řešení například chromozom $P2_2$' jedince P2. Jak je vidět z **Tab. 5**, tak tento jedinec nedostal možnost reprodukce a tím pádem by došlo ke ztrátě celého jeho genofondu, z tohoto důvodu se do procesu křížení přidává takzvaná

mutace (v **Tab. 5** naznačeno tučně), která má za následek náhodnou změnu v některém z nových jedinců [8].

Další problém je v křížení chromozomů. Pokud není vhodně zvolen způsob, jakým se jednotlivé chromozomy (veličiny) navzájem kombinují, tak se může velice nepříznivě prodloužit doba konvergence k optimu. Z tohoto důvodu se genetický algoritmus značně modifikuje v různých stadiích [8].

Výhodou genetických algoritmů je to, že i ve velice nerovnoměrně rozloženém prohledávaném prostoru dokáže nalézt globální optimum. Dokáže také pracovat se složitějšími heuristickými funkce o více parametrech. Velkou nevýhodou ale je to, že při větším množství těchto parametrů může být zapotřebí značného množství iteračních kroků a tím se prodlužuje doba potřebná k vyřešení problému. Dalším omezujícím faktorem je to, že genetický algoritmus lépe pracuje s větším množstvím jedinců v populaci a tím klade velké nároky na výpočetní techniku [8].

1.5.1 Ukázka genetického algoritmu

1.5.1.1 Kód v programu MATLAB

*FitnessFcn = @(x) -10*sin(x)./x-0.2*x-5;*	definování fitness funkce
NumberOfVariables = 1;	počet proměnných
tic	funkce pro odstartování měřiče času
[x,fval,exitFlag,Output] =	
ga(FitnessFcn,NumberOfVariables);	spuštění genetického algoritmu z knihovny
toc	funkce pro ukončení měřiče času
Output.generations	vypsání celkového počtu iteračních kroků

1.5.1.2 Vyhodnocení

Algoritmus ve všech případech nalezl maximální hodnotu zkoumané funkce. Stejně jako u simulovaného žíhání, tak i při použití tohoto algoritmu z knihovny MATLABu byla změněna znaménka. Algoritmus je totiž implicitně nastavený na minimalizaci, kdežto my potřebujeme maximalizovat. Celková doba, po kterou algoritmus hledal výsledek je 0,06s a bylo zapotřebí 51 iteračních cyklů, než algoritmus ukončil hledání. Tyto výsledky jsou pro původní nastavení parametrů algoritmu. Při změně citlivosti nebo za použití jiných metod jednotlivých kroků (způsob křížení popřípadě práce s novou generací, atd.) by se dalo dosáhnout menšího počtu kroků a tím pádem i kratší výpočetní doby.

1.5.2 MOEA (Multiple Objective Evolutionary Algorithms)

V technické praxi je mnohdy zapotřebí optimalizovat problém, který není ovlivňován pouze jedním parametrem, ale parametry více, jako je například účinnost, cena, rozměry, atd. Z tohoto důvodu vznikly více kriteriální optimalizační metody, jako jsou MOEA [9].

U všech těchto metod není možné najít jedno konkrétní optimum, ale pouze tzv. Pareto-optimální skupinu řešení. Tyto optimální řešení si lze představit tak, že jejich hodnoty heuristické (fitness)funkce odpovídají hledanému optimu, ale této hodnoty bylo dosaženo na základě různých hodnot jednotlivých parametrů. Do těchto multikriteriálních evolučních algoritmů patří například MOGA (Multiple Objective Genetic Algorithm), SPEA (Strength Pareto Evolutionary Algorithm) atd. [9].

2 SROVNÁNÍ JEDNOTLIVÝCH ALGORITMŮ

Ze všech používaných algoritmů jsem vybral jako první horolezecký algoritmus, na kterém lze snadno pochopit mechanismus optimalizace. Tento algoritmus je vhodný hlavně pro prohledávání nečlenitého prostoru, nebo pro lokální prohledávání. V opačném případě je pravděpodobné, že se algoritmus zasekne na lokálním extrému. Další fakt, který je potřeba uvážit, je silný vliv volby počátečního bodu, ze kterého algoritmus začíná hledání. Tento algoritmus jsem vyhodnotil jako nevhodný.

Jako další algoritmus jsem zvolil simulované žíhání, které přímo vychází z horolezeckého algoritmu. Tento algoritmus již počítá s pravděpodobnostmi a vyhodnocuje výhodnost dalšího kroku. Dokáže nalézt globální extrém, ale pořád je silně závislý na volbě počátečního bodu. Navíc v algoritmu není zakomponovaná žádná zpětná vazba, která by pomohla k efektivnějšímu prohledávání. Počet kroků i doba prohledávání jsou obrovské oproti ostatním algoritmům. I přesto že lze pomocí tohoto algoritmu nalézt globální optimum, tak jsem jej vyhodnotil jako nevhodný.

Následující dva algoritmy jsou ze skupiny SOAs (Swarm Optimization Algorithms) a jako první algoritmus jsem vybral včelí algoritmus. Tento algoritmus dokázal velice efektivně nalézt globální optimum zkoumané funkce. Pomocí dvou parametrů je možné ladění a lze tak algoritmus nastavit podle potřeby. Druhý algoritmus, který jsem vybral, je algoritmus mravenčích kolonií. Stejně jako u včelího algoritmu je tento algoritmus poměrně efektivní. Protože ale ani jeden z těchto algoritmů nemá dostupné knihovny v programu MATLAB, tak jsem je musel napsat vlastnoručně. Napsání univerzálního algoritmu, který by bylo možné použít nejenom na jednoduchou funkci, ale i na mnohem složitější problémy by bylo značně složité, časově náročné a navíc mimo hranice této práce. I když tyto dva algoritmy považuji za vhodné k řešení optimalizace v oblasti konstrukce elektrických strojů, tak jsem je nevybral jako výsledné kandidáty.

Posledním a nejvhodnějším kandidátem se tedy stal genetický algoritmus, který má všechny potřebné vlastnosti. Je efektivní v hledání globálního optima i v členitém prostoru. Lze jej upravit řadou způsobů tak, aby pomocí něj šly řešit i velice komplexní (multikriteriální) problémy. MATLAB obsahuje rozsáhlé knihovny a lze jej tak poměrně rychle použít i pro řešení složitějších problémů.

Název algoritmu	Doba prohledávání	Počet kroků	Vliv volby počátečního bodu	Vhodný k optimalizaci elektrických strojů
Horolezecký	(3,5 – 7) ms	64 - 306	Silný	Ne
Simulované žáhání	(187,5 – 406,3) ms	663 - 1462	Silný	Ne
Včelí kolonie	22,4 ms	12	Nulový	Ano
Mravenčí kolonie	17,4 ms	3	Nulový	Ano
Genetické	60 ms	51	Nulový	Ano

***Tab. 6** Srovnání jednotlivých algoritmů*

3 MĚŘENÍ A REALIZACE 3D MODELU MOTORU TM90-4

Z předchozích kapitol tedy vyšel nejlépe genetický algoritmus, uvážíme-li jeho aplikaci v oblasti elektrických strojů. Po vybrání vhodného algoritmu přichází na řadu jeho použití a já jsem jako cíl takovéto optimalizace zvolil motor TM90-4 od firmy EMP-Slavkov u Brna. Jedná se o 3f asynchronní čtyřpólový motor se štítkovými hodnotami viz. **Tab. 7**.

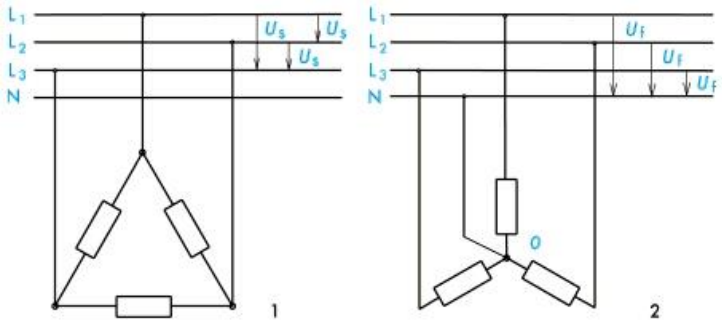
f	P_{mech}	cosφ	n_n	U (Y/D)	I (Y/D)
[Hz]	[W]	[-]	[min^{-1}]	[V]	[A]
3x50	1100	0,78	1400	400/230	2,70/4,68

***Tab. 7** Štítkové hodnoty motoru TM90-4*

Tento motor jsem zvolil z několika důvodů. Je to jeden z motorů, které jsou dostupné ve školních laboratořích, a tudíž je na něm snadno uskutečnitelné měření. Navíc je k tomuto motoru dostupná technická dokumentace, podle které je možné vytvořit 3D model.

3.1 Měření zátěžného momentu, fázových proudů a otáček

Před zahájením simulace a optimalizace jsem nejprve na motoru uskutečnil proměření některých jeho parametrů a charakteristik. Motor byl napájen z trojfázového zdroje napětí, na kterém byla nastavena hodnota 230 V, přičemž motor byl zapojen do hvězdy viz **Obr. 12**.



***Obr. 12** Trofázové zapojení 1) do trojúhelníka 2) do hvězdy [12]*

Hřídel motoru byla připojena na dynamometr, na kterém jsem postupně zvedal zatěžovací moment M_z a sledoval, jak se při tom měnily fázové proudy všech tří fází a otáčky motoru.

M_z	n	P_{mech}	I_1	I_2	I_3
[Nm]	[ot/min]	[W]	[A]	[A]	[A]
0,0	1498	0,3	1,630	1,630	1,630
0,2	1497	32,1	1,635	1,675	1,585
0,4	1495	62,9	1,635	1,675	1,585
0,6	1494	94,0	1,638	1,680	1,590
0,8	1493	125,0	1,640	1,680	1,600
1,0	1491	156,0	1,645	1,685	1,605
1,2	1490	187,0	1,650	1,695	1,605
1,4	1488	218,0	1,660	1,698	1,610
1,6	1487	248,0	1,665	1,703	1,620
1,8	1485	279,0	1,688	1,715	1,625
2,0	1483	310,0	1,695	1,725	1,638
2,2	1482	340,0	1,705	1,738	1,648
2,4	1480	371,0	1,720	1,750	1,673
2,6	1479	401,0	1,745	1,756	1,690
2,8	1477	432,0	1,768	1,775	1,705
3,0	1475	462,0	1,785	1,798	1,716
3,2	1474	492,0	1,801	1,824	1,740
3,4	1472	522,0	1,824	1,844	1,765
3,6	1470	553,0	1,860	1,863	1,785
3,8	1469	582,0	1,877	1,893	1,807
4,0	1467	612,0	1,910	1,910	1,840
4,2	1465	642,0	1,940	1,944	1,862
4,4	1463	672,0	1,962	1,973	1,890
4,6	1462	702,0	2,004	1,995	1,925
4,8	1460	731,0	2,035	2,030	1,950
5,0	1458	760,0	2,057	2,062	1,984
5,2	1456	790,0	2,101	2,084	2,022
5,4	1454	819,0	2,128	2,132	2,050
5,6	1452	848,0	2,174	2,152	2,094
5,8	1450	877,0	2,204	2,194	2,127
6,0	1448	906,0	2,248	2,230	2,158
6,2	1446	935,0	2,276	2,275	2,198
6,4	1444	963,0	2,327	2,302	2,240
6,6	1442	993,0	2,364	2,350	2,272
6,8	1439	1021,0	2,402	2,385	2,321
7,0	1437	1049,0	2,452	2,425	2,359
7,2	1435	1077,0	2,492	2,465	2,410
7,4	1432	1105,0	2,539	2,512	2,444
7,5	1431	1119,0	2,551	2,539	2,463
7,6	1430	1133,0	2,586	2,548	2,498
7,7	1428	1147,0	2,606	2,577	2,509
7,8	1427	1161,0	2,621	2,598	2,536
8,0	1425	1189,0	2,679	2,643	2,581
8,2	1422	1216,0	2,718	2,692	2,623

Tab. 8 *Naměřené hodnoty otáček n, mechanického výkobu na hřídeli P_{mech} a proudů jednotlivými fázemi v závislosti na změně zatěžovacího momentu M_z*

Z hodnot uvedených v **Tab. 8** lze vyvodit několik skutečností. S rostoucím zátěžným momentem roste také mechanický výkon a to přibližně podle vztahu:

$$P_{mech} = M_z \cdot \omega = M_z \cdot 2\pi \frac{n}{60} \tag{3.1}$$

Dále můžeme porovnat naměřené hodnoty s hodnotami odpovídajícími teoretické hodnotě zatěžovacího momentu vypočtené z štítkových hodnot v **Tab. 7**. Teoretickou velikost $M_{z\text{-}teor}$ lze vyjádřit a vypočíst opět ze vztahu 3.1. V tomto případě ale dosadíme za n a P_{mech} jmenovité štítkové hodnoty:

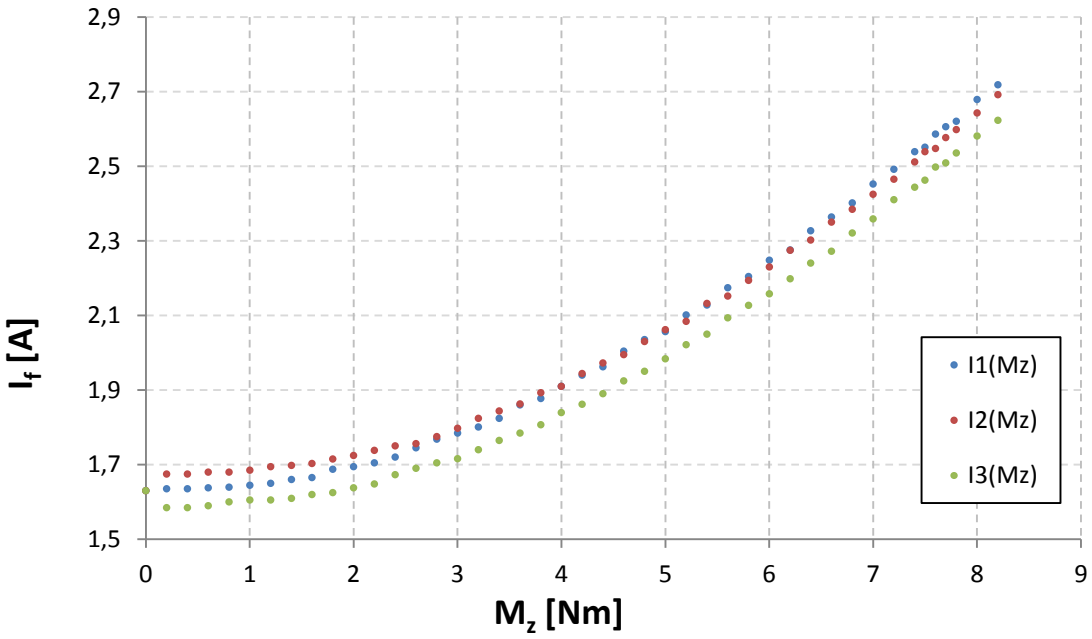
$$M_z = \frac{P_{mech}}{\omega} = \frac{P_{mech}}{2\pi \frac{n}{60}} = \frac{1100}{2\pi \frac{1400}{60}} = 7{,}5\ Nm \tag{3.2}$$

Když už tedy známe teoretickou hodnotu Mz, tak můžeme porovnat údaje udávané výrobcem se mnou změřenými hodnotami. To je nejlépe patrné z následující **Tab. 9**.

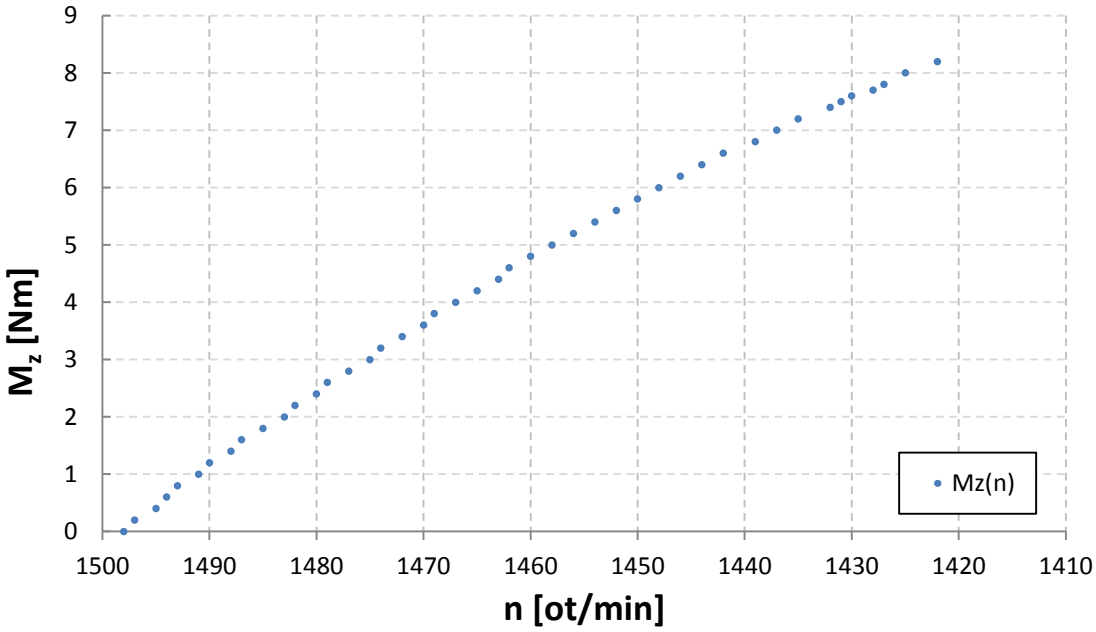
	P_{mech}	n	I_{f1}	I_{f2}	I_{f3}
	[W]	[min^{-1}]	[A]	[A]	[A]
Štítkové hodnoty	1100	1400	2,7	2,7	2,7
Z měření	1119	1431	2,551	2,539	2,463

***Tab. 9** Porovnání naměřených a vypočtených hodnot P_{mech}, n a I_f*

I v **Tab. 9**, mnohem více ale v **Tab. 8** je patrný rozdíl mezi proudy v jednotlivých fázích. Tyto rozdíly jsou způsobeny nedokonalostí skutečného motoru, ale také opotřebením materiálů, nebo jejich nehomogenitou. Z **Graf 4** je pak lépe vidět nesymetrie fázových proudů a v **Graf 5** je vynesena závislost zatěžovacího momentu na otáčkách hřídele.



***Graf 4** Závislost fázových proudů If na zátěžném momentu Mz*

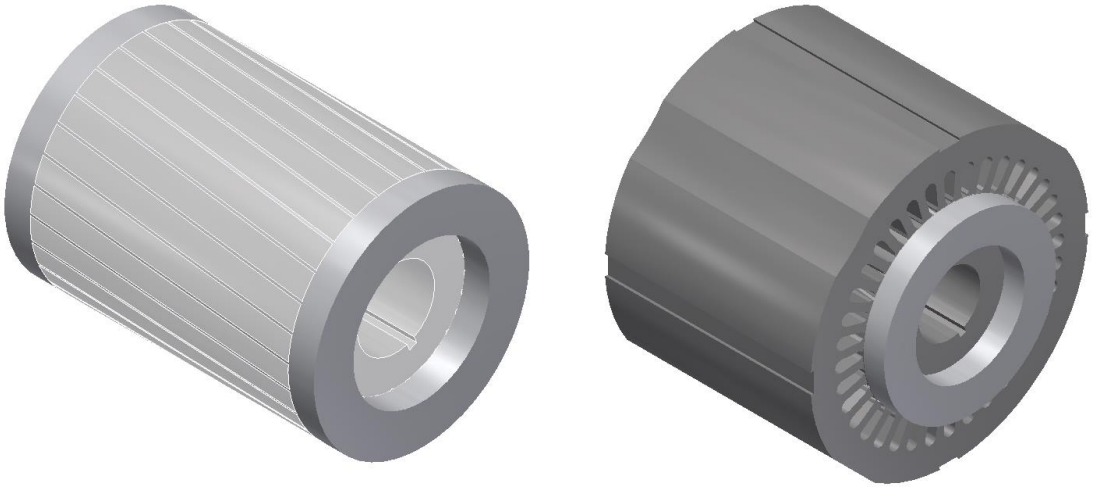


Graf 5 *Závislost zátěžného momentu M_z na otáčkách hřídele n*

Bylo by možné dále provádět teoretické výpočty a porovnávat je s naměřenými hodnotami, ale to není hlavním cílem této práce. Toto srovnání v **Tab. 9** je dostačujícím příkladem pro to, aby bylo zřejmé, že je nutné respektovat skutečné technické provedení. Je velice důležité si toto uvědomit, protože po zhotovení simulace a optimalizace je nutné správně interpretovat dosažené výsledky a uvědomit si meze jejich přesnosti.

3.2 Simulace v programu Ansoft Maxwell

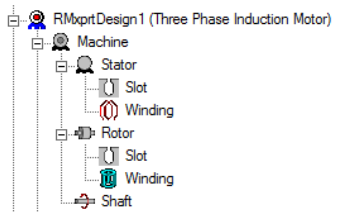
Prvotním plánem bylo vytvořit 3D model v programu Autodesk Inventor Professional (AIP). Pomocí programu Ansoft Maxwell (AM) do již existujícího modelu nechat vygenerovat statorové vinutí a nakonec takto zkompletovaný 3D model importovat do programu Ansys Workbench (AW), ve kterém by probíhala simulace a následná optimalizace. Podle plánu byl vytvořen 3D v AIP, který je vidět na **Obr. 13**.



a) Rotor s rotorovým vinutím *b) Rotor se statorem*

Obr. 13 *3D model vytvořený v AIP*

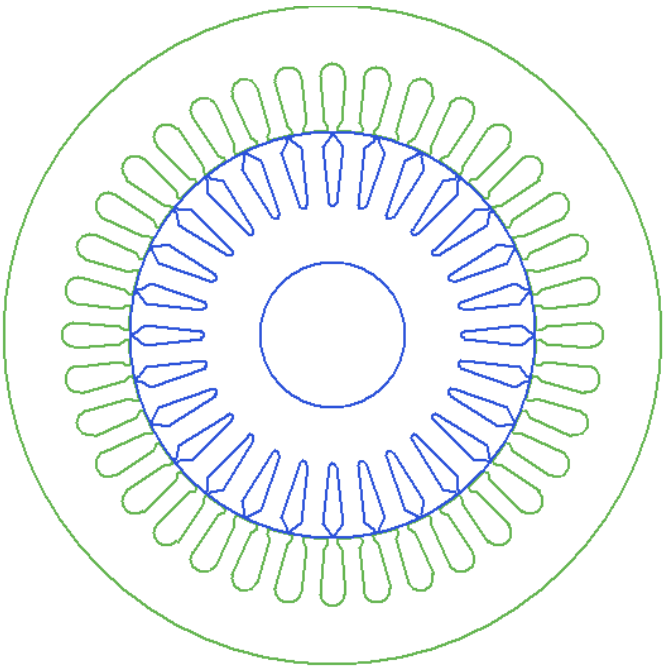
Naneštěstí jsem až po vymodelování motoru v AIP zjistil, že nové verze programu AM mají problém s importováním. Důvodů může být hned řada, ale i přes dlouhodobou snahu se import do AM nepovedl. Z toho důvodu byl původní plán upraven a kompletní model byl vygenerován již v programu AM. Tento krok sice vyřešil problémy s importem, nicméně zavedl do celé optimalizace jistou malou chybu. Tato chyba spočívá v tom, že AM generuje celý model na základě řady parametrů, které má uživatel k nastavení viz. **Obr. 14**.



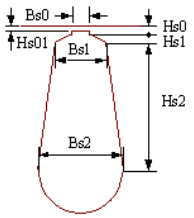
a) Uspořádání RMxprt

Name	Value	Unit	Evaluated Value
Machine Type	Three Phase Indu...		
Number of Poles	4		
Stray Loss Factor	0.01		0.01
Frictional Loss	60	W	60W
Windage Loss	20	W	20W
Reference Speed	1400	rpm	

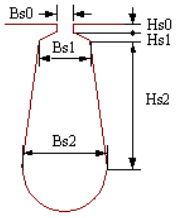
b) Zadávání parametrů



e) Vygenerovaný model motoru



c)Rotorová drážka



d) Statorová drážka

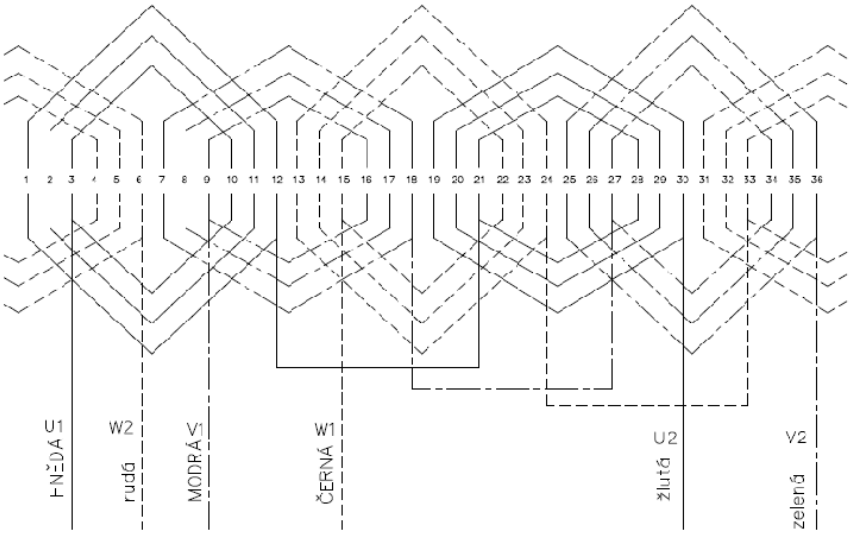
***Obr. 14** Nastavení modelu motoru v AM*

Jak je ale také patrné z **Obr. 14e**, tak už RMxprt neřeší drobné detaily, jako jsou zaoblení hran, výběžky na obvodu statoru, nebo zaoblení hran uvnitř jak statorových tak i rotorových drážek. Všechny tyto detaily sice neovlivní výsledky simulací nijak výrazně, ale je nutné je brát v potaz.

Z takto vytvořeného modelu v AM v rámci RMxprt lze vygenerovat buď 2D nebo 3D model, které již obsahují automaticky nastavené okrajové podmínky a geometrii odpovídající nastavení v RMxprt. Výhodou je, že v AM lze snadno generovat statorová vinutí viz. **Obr. 15**. Po takto nastavených cívkách statorového vinutí se při generování modelu jednotlivé cívky přímo vkládají do statorových drážek. Směry proudů protékaných těmito cívkami jsou určeny automaticky podle **Obr. 15a**.

Protože výpočty v simulaci 3D modelu jsou velice náročné na výpočetní výkon a je téměř nemožné je provádět na stolních počítačích a dodržet při tom rozumný počet kroků, tak jsem pro potřeby srovnání výsledků simulace s mými naměřenými hodnotami zvolil pouze 2D model. Simulaci jsem nastavil pro časový úsek 0 až 0,14s a to odpovídá 7 periodám síťové frekvence. Krok, se kterým probíhaly výpočty, jsem nastavil na 0,002s. Takto nastavená simulace mého 2D modelu trvala přibližně 25 minut. Při pokusech o 3D simulaci trval výpočet jednoho kroku asi hodinu.

	Phase	Turns	In Slot	Out Slot
Coil_1	A	110	3	10
Coil_2	A	110	2	11
Coil_3	A	110	1	12
Coil_4	A	110	21	28
Coil_5	A	110	20	29
Coil_6	A	110	19	30
Coil_7	B	110	9	16
Coil_8	B	110	8	17
Coil_9	B	110	7	18
Coil_10	B	110	27	34
Coil_11	B	110	26	35
Coil_12	B	110	25	36
Coil_13	C	110	15	22
Coil_14	C	110	14	23
Coil_15	C	110	13	24
Coil_16	C	110	33	4
Coil_17	C	110	32	5
Coil_18	C	110	31	6



a) Zapojení v RMxprt *b) Schéma zapojení 4 pólového statorového vinutí*

***Obr. 15** Zapojení statorového vinutí*

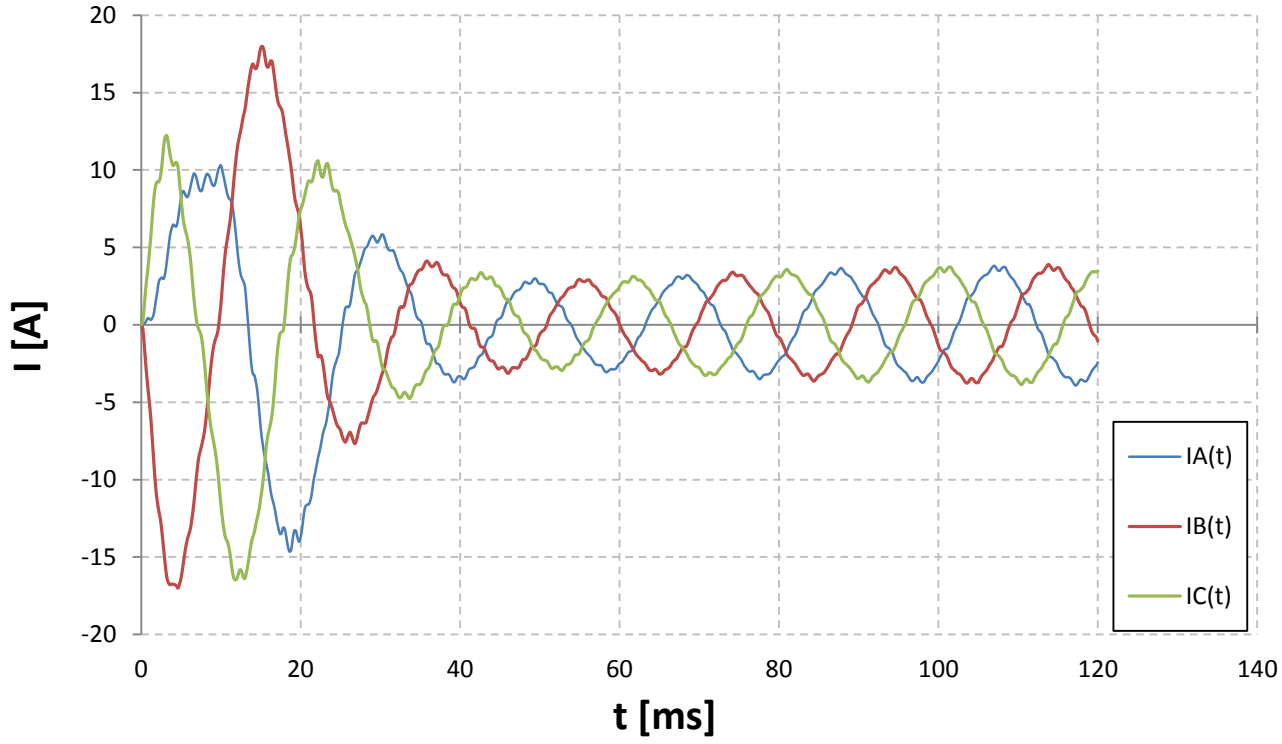
3.3 Srovnání výsledků simulace a měření

Než bude možné přistoupit k optimalizaci, tak je nutné mít k dispozici informaci o důvěryhodnosti simulace. Nejlepším srovnáním bude doplnění **Tab. 9** o výsledky simulace.

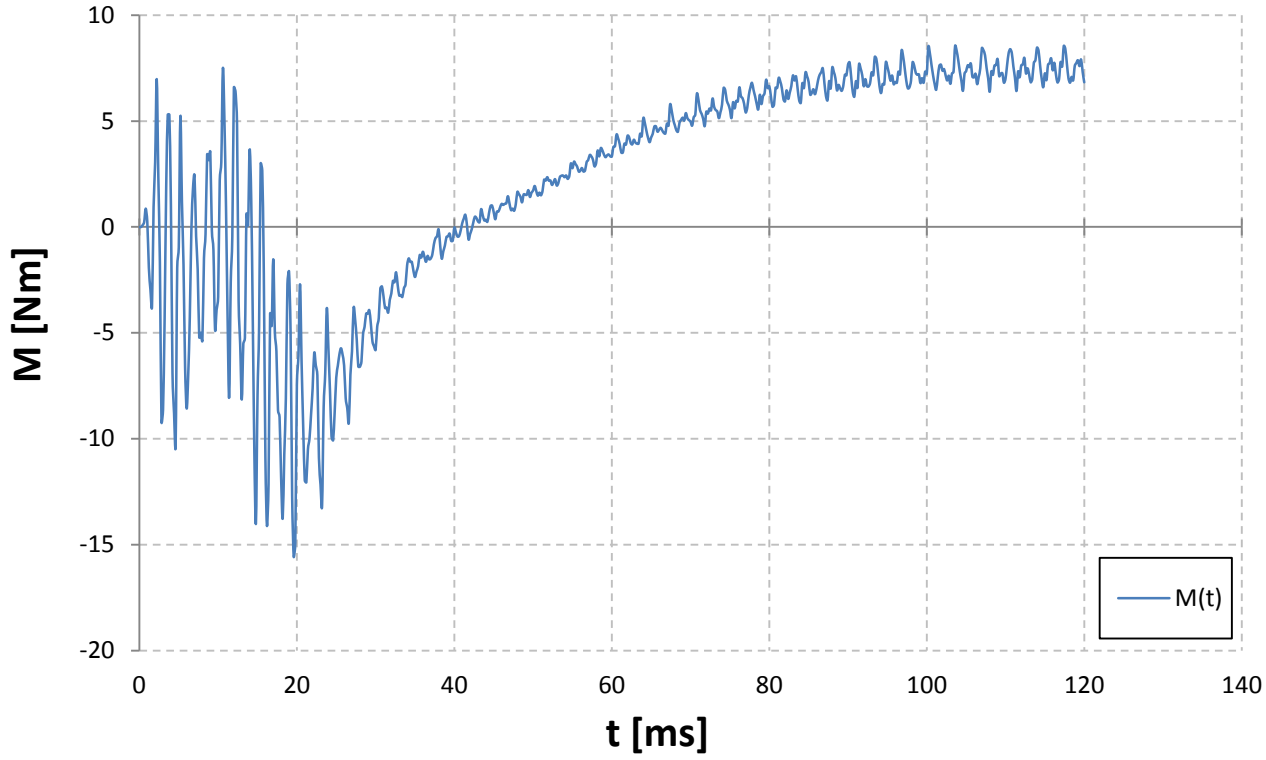
	P_{mech}	n	I_{f1}	I_{f2}	I_{f3}
	[W]	[min^{-1}]	[A]	[A]	[A]
Štítkové hodnoty	1100	1400	2,7	2,7	2,7
Z měření	1119	1431	2,551	2,539	2,463
Ze simulace	1169	1450	2,864	2,839	2,865

***Tab. 10** Doplnění **Tab. 9** o výsledky simulace*

Jak je vidět ze srovnání hodnot, tak jsou výsledky simulace o několik procent větší než naměřené hodnoty. Je zajímavé, že i v simulaci jsou hodnoty fázových proudů od sebe odlišné. V tomto případě ale jde o chybu, kterou by šlo odstranit statistickým zpracováním výsledků. Jak je vidět v **Graf 6**, tak přibližně od čtvrté periody jsou již proudy ustálené, bylo by tedy možné zprůměrovat amplitudy v jednotlivých půlperiodách a při dostatečně dlouhém časovém vzorku by se jak amplitudy, tak i efektivní hodnoty proudů v jednotlivých fázích blížily ke stejné hodnotě. Důležitý rozdíl je ale u momentu viz **Graf 7**. Zprůměrujeme-li hodnoty v ustálené části, řekněme mezi 100 ms a 120 ms, tak obdržíme hodnotu 7,7 Nm, nikoli 7,5. Naměřené hodnoty pro 7,7 Nm z **Tab. 8** se už simulaci přibližují více. To, že hodnota momentu je mírně vyšší, než u skutečného motoru je nejpravděpodobněji způsobeno tím, že reálný motor má větší ztráty než byly zadány do modelu v RMxprt viz **Obr. 14a**. Bylo by možné v laboratoři provést měření naprázdno a nakrátko, a z hodnot získaných tímto měřením se dopočítat přesnějším hodnotám ztrát. S přesnějšími a podrobnějšími hodnotami by se pak měření a simulace shodovaly mnohem více. Kvantitativně se simulace liší od údajů udávaných výrobcem o 4% až 6%. Tuto odchylku je tedy nutné respektovat i u následné optimalizace.



Graf 6 *Simulovaná závislost fázových proudů na čase*



Graf 7 *Simulovaná závislost momentu na čase*

4 OPTIMALIZACE

Jak je vidět na **Graf 6**, tak v okamžiku připojení motoru k síti začne procházet statorovým vinutím značně velký rozběhový proud. Velikost rozběhového proudu určená simulací je 5,6 násobkem proudu jmenovitého. Takto silný proudový impulz vytváří silné statorové magnetické pole, jehož působením se v rotorovém vinutí indukují také vysoké proudy. Výsledné silové účinky těchto dvou magnetických polí je možné pozorovat prostřednictvím momentu předávaného na hřídel viz **Graf 7**. Protože takto velké silové účinky jsou nepříznivé pro životnost motoru, tak by bylo dobré zjistit, zda je možné je nějakým způsobem omezit. Jak již bylo řečeno, tak se vše odvíjí od velikosti rozběhového proudu a velikost tohoto proudu je do značné míry ovlivněna materiálem rotorového vinutí a tvarem rotorových drážek [13].

Tato problematika byla dříve řešena různými způsoby, jako je například typ rotorových klecí - dvojitá nebo vírová. Protože změna materiálu nebo kombinace více klecí z různých materiálů je náročná na výrobu je tedy i nákladná. Lze ale použít novějšího typu klecí, jako jsou například vírové. Tvarem rotorových drážek lze tedy významně měnit velikost rozběhového proudu a zároveň je to vhodný příklad pro optimalizaci.

Mnou vytvořený 3D model v programu AM jsem tedy importoval do programu AW. V programu AW je zahrnuta řada možných výpočetních metod a to včetně optimalizačního genetického algoritmu, který mi ze všech optimalizačních metod vyhověl nejvíce. Celý proces probíhá ve třech krocích, a to sice vygenerování náhodných vstupních parametrů, vypočtení výstupních parametrů pro každou kombinaci vstupů a nakonec optimalizace, jejíž podrobnější popis je v kapitole 1.5.

Jako vstupní parametry jsem zvolil čtyři hlavní rozměry rotorové drážky a to sice B_{s0}, B_{s1}, B_{s2} a H_{s2}. Tyto parametry jsou vyobrazeny na **Obr. 14c.** Optimalizaci jsem nastavil tak, že v prvním kroku bylo vygenerováno 25 různých kombinací těchto vstupních parametrů. V druhém kroku pro každou kombinaci proběhla simulace, ze které byly zjištěny parametry výstupní, jako které jsem zvolil maximální proud v každé z fází (ty odpovídají rozběhovému proudu) a moment na hřídeli. V poslední řadě proběhla optimalizace, která ze všech 25 simulovaných variant vypočetla 3 nejlepší kandidáty. Tito kandidáti jsou výsledkem, který odpovídá tomu, jak jsem optimalizaci nastavil kritéria hledání. Cílem bylo minimalizovat rozběhový proud a zároveň udržet moment co možná nejblíže hodnotě 7,5Nm (jmenovitý moment).

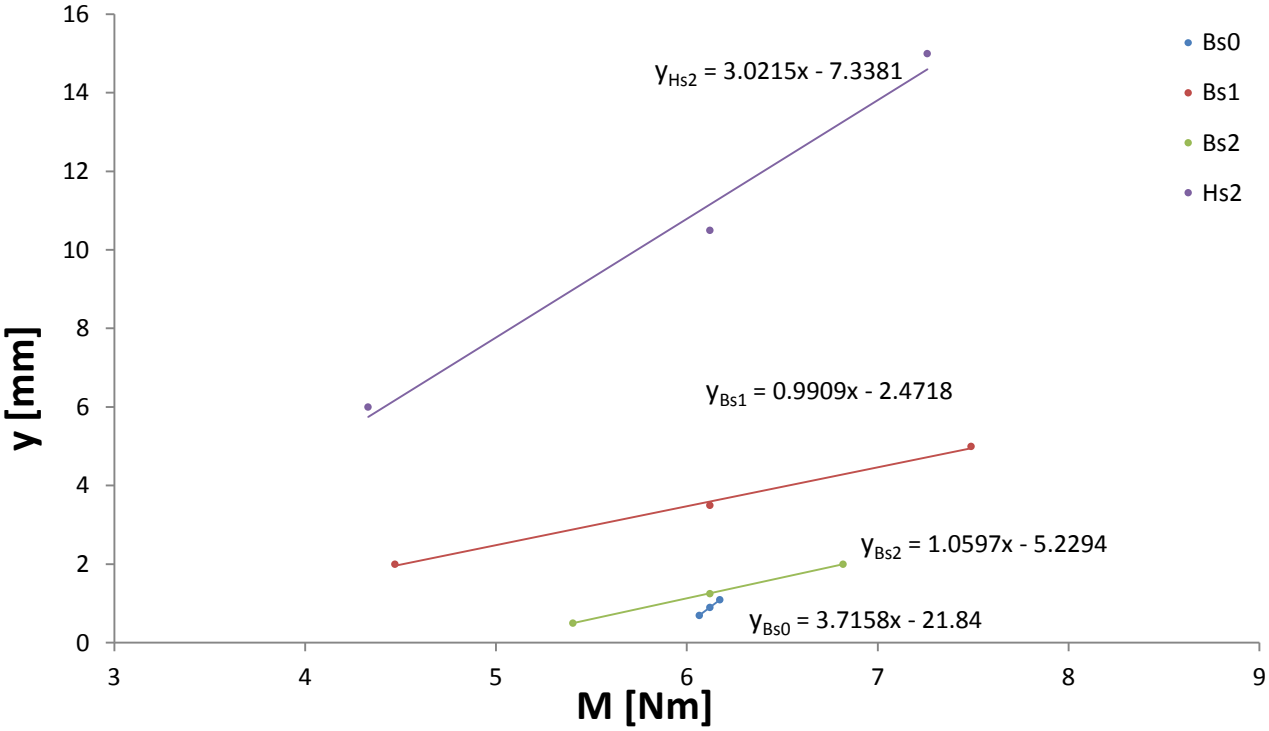
4.1 Výsledky optimalizace

V **Tab. 11** jsou vidět všechny 3 kroky popisované v předchozí kapitole a zároveň i srovnání s hodnotami původní simulace. Je vidět jakým způsobem byly kombinovány vstupní parametry a jaký to mělo důsledek na výstupní parametry. Porovnáním optimalizovaných a původních hodnot jsem zjistil, že algoritmus nedokázal nalézt takovou kombinaci vstupních parametrů, aby výrazně zmenšil rozběhový proud. To je velice kladný výsledek, protože na jeho základě usuzuji, že tvar drážek byl navržen správně. Rozměry jsou ale v původním a optimalizovaném tvaru rozdílné. Příčinu těchto rozdílu vidím v nedostatečném počtu parametrů a to jak vstupních tak i výstupních. Protože rozběhový proud i moment jsou téměř totožné, tak bych rozdíly v rozměrech drážek přisoudil nějakému jinému parametru, který nebyl v optimalizaci zahrnut.

číslo simulace	B_{s0} [mm]	B_{s1} [mm]	B_{s2} [mm]	H_{s2} [mm]	I_{Amax} [A]	I_{Bmax} [A]	I_{Cmax} [A]	M [Nm]
1	0.90	3.50	1.25	10.50	9.40	16.16	11.67	6.12
2	0.70	3.50	1.25	10.50	9.23	15.93	11.48	6.07
3	1.10	3.50	1.25	10.50	9.56	16.38	11.86	6.17
4	0.90	2.00	1.25	10.50	8.01	13.02	10.25	4.47
5	0.90	5.00	1.25	10.50	10.45	18.33	12.63	7.49
6	0.90	3.50	0.50	10.50	8.94	15.23	11.56	5.40
7	0.90	3.50	2.00	10.50	9.80	16.97	11.76	6.82
8	0.90	3.50	1.25	6.00	8.08	13.45	11.35	4.33
9	0.90	3.50	1.25	15.00	10.11	17.53	11.76	7.26
10	0.76	2.44	0.72	7.33	7.07	11.28	10.26	3.45
11	1.04	2.44	0.72	7.33	7.28	11.59	10.59	3.52
12	0.76	4.56	0.72	7.33	8.95	15.36	12.00	5.34
13	1.04	4.56	0.72	7.33	9.17	15.64	12.22	5.41
14	0.76	2.44	1.78	7.33	7.91	13.13	10.56	4.40
15	1.04	2.44	1.78	7.33	8.15	13.46	10.89	4.47
16	0.76	4.56	1.78	7.33	9.51	16.51	12.16	6.14
17	1.04	4.56	1.78	7.33	9.72	16.80	12.39	6.21
18	0.76	2.44	0.72	13.67	8.49	14.22	10.56	5.17
19	1.04	2.44	0.72	13.67	8.75	14.56	10.89	5.24
20	0.76	4.56	0.72	13.67	10.39	18.23	12.30	7.63
21	1.04	4.56	0.72	13.67	10.57	18.51	12.53	7.73
22	0.76	2.44	1.78	13.67	9.15	15.59	10.70	6.31
23	1.04	2.44	1.78	13.67	9.41	15.95	11.03	6.40
24	0.76	4.56	1.78	13.67	10.78	19.03	12.37	8.37
25	1.04	4.56	1.78	13.67	10.95	19.31	12.60	8.51
pořadí	výsledky optimalizace							
1	0.90	3.63	1.30	14.76	10.20	17.77	11.86	7.42
2	0.83	3.81	1.51	14.34	10.33	18.08	11.94	7.67
3	0.77	3.99	1.51	13.48	10.31	18.13	12.01	7.65
pořadí	původní simulace							
1	0.90	4.20	1.60	11.60	10.30	17.99	12.22	7.44

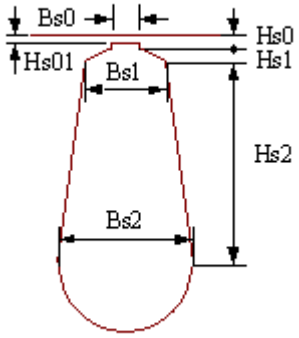
***Tab. 11** Výsledky optimalizace*

Velmi zajímavý je také rozsah, v jakém se pohybovaly hodnoty momentu $M \in \langle 3{,}45\,;\,8{,}51 \rangle$Nm. Je tedy zřejmé, že vliv tvaru drážek má významný dopad na moment resp. proud. Mimo srovnání původních a optimalizovaných rozměrů bylo navíc zjištěno, jak velký mají jednotlivé rozměry vliv na moment motoru. Ideální by bylo změnit nastavení optimalizace a provést čtyři samostatné optimalizace, pokaždé pro jiný z vstupních parametrů. To by ovšem bylo časově náročné, a proto jsem se přiklonil k méně přesné, zato rychlejší variantě. V **Tab. 11** se vždy vyskytují alespoň tři kombinace, u kterých se mění pouze jeden z parametrů. Sestrojil jsem tedy **Graf 8**, ve kterém jsou čtyři skupiny po třech bodech. Proložíme-li tyto trojice bodů, můžeme odečíst jednotlivé směrnice pro každou trojici bodů.



Graf 8 *Závislost momentu na rozměrech statorové drážky*

Z velikosti těchto směrnic jsem usoudil, že moment motoru je nejvíce ovlivňován rozměry B_{s1} a B_{s2}, o něco méně je pak ovlivněn parametrem H_{s2} a v poslední řadě parametrem B_{s0}, který má nejmenší vliv.



Obr. 16 *Rotorová drážka*

5 ZÁVĚR

V této práci jsem se zabýval popisem vybraných optimalizačních metod použitelných mimo jiné i v oblasti elektrotechniky. Protože optimalizace elektrických strojů je komplexní problém s mnoha proměnnými, tak jsem tyto mnou vybrané algoritmy otestoval na jednoduché funkci, abych zjistil, zda jsou nebo nejsou vhodné k tomuto druhu optimalizace. Jako nejvhodnější jsem vybral genetický algoritmus. Ten totiž dokáže pracovat s velkým množstvím parametrů a hodí se tak na optimalizaci složitějších problémů, jako je elektrický motor.

Dále jsem vybral typ motoru, na kterém bych mohl vyzkoušet vybraný algoritmus. Zvolil jsem 3f asynchronní čtyřpólový motor TM90 – 4 od firmy EMP-Slavkov u Brna. Volba tohoto motoru je dána tím, že motor je k dispozici ve školních laboratořích a je známá jeho konstrukce. Díky tomu jsem mohl vytvořit 3D model v programu AM v prostředí RMXprt, ze kterého jsem nechal vytvořit plnohodnotný 2D model se všemi okrajovými podmínkami a parametry potřebnými k zahájení simulace. Abych mohl objektivně posoudit výsledky simulace, tak jsem provedl měření.

Prvním významným zjištěním této práce je už jen samotné porovnání štítkových, naměřených a simulovaných hodnot. Porovnání je v **Tab. 10**. Srovnání těchto hodnot je důležité, protože je z něj vidět, do jaké míry se odlišuje simulace od skutečných naměřených parametrů, ale zároveň je patrné, jak se měření liší od hodnot udávaných výrobcem. Vzájemné odchylky jsou v rozmezí od 4% do 6%, a to je relativně dobrý výsledek. Lze tedy očekávat, že výsledky optimalizace budou zatíženy obdobnou odchylkou.

Ve zhotoveném 2D modelu jsem po provedení prvotní simulace nastavil sadu vstupních parametrů, za které jsem zvolil rozměry rotorové drážky. Jak je podrobněji popsáno v kapitole 4, tak změnou těchto rozměrů lze značně ovlivňovat vlastnosti motoru. Dále jsem nastavil, jaké mají být výstupní parametry a v jakém časovém intervalu se mají provádět výpočty. Sledoval jsem nejvýznamnější dvě veličiny a to maximální hodnotu fázových proudů (ty odpovídají rozběhovému proudu, viz **Graf 6**) a průměrný moment motoru na hřídeli v intervalu od 120ms do 140ms (v tomto intervalu je již moment ustálený, viz **Graf 7**).

Takto nastavený 2D model jsem z AM importoval do AW, ve kterém jsem k modelu přiřadil optimalizační metodu. Použil jsem rozšířenou verzi GA, a to sice algoritmus se zkratkou MOGA (Multi Objective Genetic Algorithm). Tato upravená verze je vhodnější pro optimalizace více parametrů. Po spuštění optimalizace se uskutečnilo několik kroků podrobněji popsaných v kapitole 4.1. Celý proces trval přibližně 20 hodin čistého času, což značně komplikuje postupné ladění nastavení optimalizačního parametru. Takovýchto 20 hodinových cyklů jsem zopakoval několik, než se mi povedlo odstranit všechny nesrovnalosti a výsledky této poslední optimalizace jsou v **Tab. 11**. Úkolem optimalizace bylo zmenšit rozběhový proud změnou rozměrů rotorových drážek při zachování momentu na jmenovité hodnotě 7,5Nm.

Stěžejním a nejdůležitějším výsledkem této práce je porovnání výsledků optimalizace s původními hodnotami. Z tohoto porovnání, které je ve spodní části **Tab. 11** jsem vyvodil několik závěrů. Prvním a nejvýznamnějším je to, že hodnoty optimalizovaných proudů a momentu jsou téměř totožné s původními hodnotami před optimalizací. To značí, že rotorové drážky původního motoru jsou navrženy dobře. Co už ale není stejné, jsou některé rozměry drážek. U rozměru Bs2 a Hs2 se rozměry liší o 19% a 27%. Z těchto odlišností usuzuji, že se při navrhování drážek uvažují některé další výstupní veličiny, které ale nebyly zahrnuty v mé optimalizaci.

Cílem této práce bylo navrhnout optimalizační algoritmus a aplikovat jej na jednoduchý příklad, což bylo uskutečněno s kladnými výsledky. Při tvoření této práce jsem ale zjistil, jak je proces navrhování motoru složitý. Nelze uvažovat pouze některé hlavní veličiny, ale je nutno vzít v potaz i drobné detaily. V budoucnu by bylo možné tuto práci rozšířit o podrobnější měření a zahrnout do optimalizace více proměnných. S výhodou lze do již před chystaného postupu přidat další prvky nebo dokonce i změnit typ motoru. Lze libovolně směřovat optimalizaci na různé části motoru a na základě toho pozorovat jak, různé konstrukční prvky ovlivní chod motoru. Myslím, že v budoucnu by bylo možné na takovémto nebo podobném principu založit metodu kompletního návrhu motoru.

LITERATURA

[1] *Multikriteriální optimalizace asynchronního stroje* [online]. [cit. 2015-10-4]. Dostupné z: http://www.elektrorevue.cz/clanky/02055/index.html

[2] *Profi elektrika.cz* [online]. [cit. 2015-10-4]. Dostupné z: http://elektrika.cz/data/clanky/stoleti-plne-vynalezu

[3] *YouTube:NMCS4ALL: Optimization by Hill climbing* [online]. [cit. 2015-10-11]. Dostupné z: https://www.youtube.com/watch?v=boTeFM-CVFw&list=PLQ1QR4aj_yuZtzHl_Egie4HM6DllzXIpW

[4] *YouTube: Mod-01 Lec-10 Hill Climbing* [online]. [cit. 2015-10-11]. Dostupné z: https://www.youtube.com/watch?v=ZOvRZ7UJMjk

[5] http://www.kod.tul.cz/predmety/PSI/Prednasky/prednasky_2015/prednaska_2014_5.pdf

[6] *YouTube: Mod-01 Lec-14Optimization 1 (Simulated Annealing)* [online]. [cit. 2015-10-20]. Dostupné z: https://www.youtube.com/watch?v=dg5zUxdAE_E&index=14&list=PLbMVogVj5nJQu5qwm-HmJgjmeGhsErvXD

[7] *Insects: The Bees Algorithm* [online]. [cit. 2015-10-24]. Dostupné z: https://orca-mwe.cf.ac.uk/53653/1/Yuce%202013.pdf

[8] *YouTube: Mod-01 Lec-1514 Optimization 2 (Genetic Algorithms)* [online]. [cit. 2015-10-25]. Dostupné z: https://www.youtube.com/watch?v=THSNf9mPsmA&index=15&list=PLbMVogVj5nJQu5qwm-HmJgjmeGhsErvXD

[9] *Diva-portal: Multi-Objective Optimization using Genetic Algorithms* [online]. [cit 2015 -10-30] Dostupné z: http://www.diva-portal.org/smash/get/diva2:570751/FULLTEXT01.pdf

[10] COELLO COELLO, Carlos A, David A VAN VELDHUIZEN a Gary B LAMONT. *Evolutionary algorithms for solving multi-objective problems*. New York: Kluwer Academic, c2002, xxxiii, 576 p. ISBN 0306467623.

[11] *MathWorks: Optimize Using Simulated Annealing* [online]. [cit. 2015-11-18]. Dostupné z: http://www.mathworks.com/help/gads/performing-a-simulated-annealing-optimization.html

[12] *Loporelo.info:trojfázová zapojení* [online]. [cit. 2016-4-10]. Dostupné z: https://leporelo.info/zapojeni-trojfazova

[13] ONDRŮŠEK, Čestmír. *Skripta Elektrických strojů*[online]. Brno [cit. 2016-04-28]. *Zveřejněno na E-lerningu jako studijní materiál k předmětu BESB*